# Security-Driven Software Evolution Using A Model Driven Approach

## PhD Thesis

**Hui Guan**

Software Technology Research Laboratory

De Montfort University

2014

To my husband, Ming Ye,
my son, Jihua Ye and
my families.

# Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University, UK, from January 2009 to April 2014. Apart from the degree that this thesis is currently applying for, no other academic degree or award was applied for by me based on this work.

# Acknowledgements

Without the support and aid from many people, I would not complete my PhD study. I'd like to take the opportunity to thank all of them.

Above all, I would like express my deepest gratitude towards my first supervisor, Prof. Hongji Yang, for his excellent guidance, invaluable advice, patience, encouragement and providing me with an excellent research atmosphere during my PhD study. My research career benefits tremendously from the research methodologies he introduced to me. I also reserve indebted for his support and encouragement during the times when I was ill and depressed.

My heartfelt thanks are remained for my second supervisor, Prof. Weiru Chen, for his deep insights and invaluable suggestions. I really appreciate his support and encouragement both on my work and on my PhD study. Without his help, I can't concentrate on my final year's writing up.

I wish to thank Dr. Xiaodong Liu and Dr. Feng Chen for their carefully examining the thesis and the invaluable comments they gave to me.

I would like to take this opportunity to thank the members of Creative Computing Laboratory of Bath Spa University: Prof. Andrew Hugill, Prof. Ron George and Mrs. Julie Neale, for providing me such wonderful research environment when I complete my thesis there. Also, my thanks go to Charles Stuart Edwards for his kind help when I debugged the case stufy.

Especially, I wish to express thanks to my husband Ming Ye, for his unconditional love, patience, financial and emotional support, I would not make it this far without him. I also thank my passed away parents, my parents in law, my sisters and brother, for all their love, care, encouragements and supports over the years. Special thanks to my son, Jihua Ye, for his understanding and support during my absence for PhD study. This thesis is dedicated to them.

Finally, I would like to acknowledge all of the colleagues in Software Technology

Research Laboratory at De Montfort University, Creative Computing Centre at Bath Spa University and the colleagues in Shenyang University of Chemical Technology, for their valuable suggestions and encouragement, for their understanding and support, including: Dr. Feng Chen, Dr. Runjie Liu, Dr. Shang Zheng, Dr. Jiantao Zhou, Dr. Li Li, Prof. Jun Liu and many others. I am indebted to all of them.

# Abstract

High security level must be guaranteed in applications in order to mitigate risks during the deployment of information systems in open network environments. However, a significant number of legacy systems remain in use which poses security risks to the enterprise' assets due to the poor technologies used and lack of security concerns when they were in design. Software reengineering is a way out to improve their security levels in a systematic way. Model driven is an approach in which model as defined by its type directs the execution of the process. The aim of this research is to explore how model driven approach can facilitate the software reengineering driven by security demand. The research in this thesis involves the following three phases.

Firstly, legacy system understanding is performed using reverse engineering techniques. Task of this phase is to reverse engineer legacy system into UML models, partition the legacy system into subsystems with the help of model slicing technique and detect existing security mechanisms to determine whether or not the provided security in the legacy system satisfies the user's security objectives.

Secondly, security requirements are elicited using risk analysis method. It is the process of analysing key aspects of the legacy systems in terms of security. A new risk assessment method, taking consideration of asset, threat and vulnerability, is proposed and used to elicit the security requirements which will generate the detailed security requirements in the specific format to direct the subsequent security enhancement.

Finally, security enhancement for the system is performed using the proposed ontology based security pattern approach. It is the stage that security patterns derived from security expertise and fulfilling the elicited security requirements are selected and integrated in the legacy system models with the help of the proposed security ontology.

The proposed approach is evaluated by the selected case study. Based on the analysis, conclusions are drawn and future research is discussed at the end of this thesis. The results show this thesis contributes an effective, reusable and suitable evolution approach for software security.

# Table of Contents

# List of Figures

List of Figures

List of Figures

# List of Tables

List of Tables

# List of Lists

# List of Acronyms

| | |
|---|---|
| *AES* | *Advanced Encryption Standard* |
| *AOM* | *Aspect-Oriented Modelling* |
| *AOP* | *Aspect-Oriented Programming* |
| *AOSD* | *Aspect-Oriented Software Development* |
| *AST* | *Abstract Syntax Tree* |
| *CERT* | *Computer Emergency Readiness Team* |
| *CIM* | *Computational Independent Model* |
| *CLASP* | *Comprehensive Lightweight Application Security Process* |
| *COTS* | *Commercial Off The Shell* |
| *CSDG* | *Class Scenario Dependency Graph* |
| *CVSS* | *Common Vulnerability Scoring System* |
| *DES* | *Data Encryption Standard* |
| *DFD* | *Data Flow Diagram* |
| *J2EE* | *Java 2 Platform Enterprise Edition* |
| *MDA* | *Model Driven Architecture* |
| *MDD* | *Model Driven Development* |
| *MDE* | *Model Driven Engineering* |
| *MDS* | *Model Driven Security* |
| *MOF* | *Meta-Object Facility* |
| *MVC* | *Model-View-Controller* |
| *NFR* | *Non-Functional Requirement* |

List of Acronyms

| | |
|---|---|
| *NIST* | *National Institute of Standards and Technology* |
| *PIM* | *Platform Independent Model* |
| *PSM* | *Platform Specific Model* |
| *OCTAVE* | *Operationally Critical Threat, Asset, and Vulnerability Evaluation* |
| *OMG* | *Object Management Group* |
| *OWASP* | *Open Web Application Security Project* |
| *OWL* | *Web Ontology Language* |
| *RBAC* | *Role Based Access Control* |
| *RDF* | *Resource Description Framework* |
| *SDL* | *Security Development Lifecycle* |
| *SEMDA* | *Security-driven software Evolution using Model Driven Approach* |
| *SOA* | *Service-Oriented Architecture* |
| *SSL* | *Secure Socket Layer* |
| *UML* | *Unified Modelling Language* |
| *W3C* | *World Wide Web Consortium* |
| *XMI* | *XML Meta-data Interchange* |
| *XML* | *eXtensible Markup Language* |

# Chapter 1

# Introduction

**Objectives**

- To motivate the need for security driven software evolution
- To describe the research objectives and research methods
- To highlight original contribution and define the success criteria
- To outline the structure of the thesis

## 1.1 Motivation

With the rapid development of IT technology, more and more enterprises are likely to connect their software systems to the internet where the enterprises' assets are exposed to lots of various potential damages, which is usually referred to as electronic risks [8, 51]. High security levels must be guaranteed in applications so as to reduce the risks and facilitate the deployment of information systems in open network environments. However, because of the critical value to the business process, there are a significant number of legacy systems are kept in operation. Most of them cannot be connected to the open network directly because they were originally designed to be used in protected environment and thereby the security features provided in their designs are not enough to protect them against the security attacks in the open network environment. Obviously, these applications cannot operate properly in modern internet environment. Nevertheless, these legacy systems are vital to the enterprise in that they are tightly coupled with the enterprise's information and production infrastructure and have been thoroughly tested during many years operation.

Statistics derived from the Software Engineering Institute's CERT (Computer

Emergency Readiness Team) Coordination Centre, a centre of Internet security expertise, show that vulnerabilities in software are currently at a rate of over 4,000 per year [23]. A number of surveys report significant financial losses due to attacks exploiting [24]. With the advent of the Internet, many of these existing systems are now in the open vulnerable environments. Hence the risks from malicious attacks are increasing. Some of the main causes presented in [112] and [168] can be used to explain the development, such as increased network connectivity, easily extensible, the complexity and huge size of code, easier to attack tools and attack sophistication. Many security methodologies and countermeasures have been developed to improve software security during various stages of the software development lifecycle. However, most of the existing approaches are limited to the integration security from the scratch of software development in the new designed software systems. Several approaches [110, 168] have been proposed to address the security problems. However, theses approaches mainly focus either on the handling of security in a specific stage during the software development such as requirement, design phase or on specific security properties such as authentication, access control. Moreover, most of them are implemented at the late stage of software development life cylce so that it will cost more to find and remove the security problems.

Information technology tends to focus on new systems - the processes for designing, developing, testing, and employing. Making them secure has been the subject of thousands of books and the focus of hundreds of processes. Building new systems is high-profile, difficult work that receives appropriate attention, but IT operations of an organisation rely most heavily on systems that are already in place - the legacy systems. Legacy systems make up the vast bulk of the code base, and all new systems become legacy when they come on line [122].

The security of legacy systems, then, are as important as that of new systems because they [122]:

- are central the organisation’s operation
- make up the bulk of the code base
- provide a point of entry to the enterprise

- have changed more than most of the organisation realises
- often are changed without the scrutiny as those given to new systems
- are impacted by changes in external systems, services, and infrastructure
- involve old components and technology that may never have been tested

Undoubtedly, more and more existing software will become to legacy ones due to the rapid development of information technology and constantly changing requirements. It is foreseeable that in the years to come there will be an increasing demand for perfective maintenance actions aiming at improving the security level of existing applications [29]. Therefore, it is necessary for a systmatic approach to evolve the software system with more security features.

Software evolution [14, 183] is a way out. Being a broader name to software maintenance, software evolution refers to the study and management of the process of making changes to software over time which comprises development activities, maintenance activities and a sequence of software reengineering activities. Software reengineering [14] is the core technique for successful software evolution which can be seen as a combination of reverse engineering, functional restructuring and forward engineering. The purpose of software reengineering is to utilise the advantages of existing systems and new technologies. On one hand, existing systems can be reengineered by taking advantages of new technologies, while on the other hand; new development efforts derive benefit from reusing existing system. Software productivity and quality can be improved by using software reengineering across the whole life cycle [25].

The typical software reengineering techniques focus on the process starting from program source code which is thought as the only reliable information in a legacy system. During the reengineering, the program specifications can be recovered from the legacy source code and will then facilitate legacy system's migration in the following forward engineering phase. There is a trend in recent researches in reengineering domain that ideas in the context of Model Driven Architecture (MDA) [137] may give light to the reengineering research. MDA aims at a unified model based architecture for software development. All software artefacts, including requirement specifications,

architecture and design descriptions, and even code, are regarded as models and represented by modelling languages. A series of models form a hierarchical abstraction levels and models of MDA at various abstraction levels can be transformed automatically to each other.

Up to now, there is not enough attention for combining traditional software reengineering techniques and the software security researches, which is actually of significant importance for security improvement for legacy system. This situation leads to an increasing requirement to carry out security evolution more efficiently, which triggers the research described in this thesis. The thesis therefore aims to present an approach to reengineering the legacy software systems for security concerns in line with MDA philosophy.

## 1.2 Research Objectives

The proposed research aims to implement a software reengineering focusing on the security concerns during software evolution which will bridge the gap between software security and software evolution with the following objectives:

- To provide a systematic approach to performing security - driven software evolution
- To create a guideline for reverse engineering security based on models
- To develop a risk assessment method to elicit security requirements for web applications
- To present a security enhancement method to fulfil the identified security requirements
- To deploy and validate the proposed approach in a legacy system

The proposed research is comprehensive, which covers redevelopment in the software reengineering process; it is constructive, which develops a new theory, algorithms, models, strategies or methods. However, complicated interaction between human activities and the software system cannot be avoided in the research.

## 1.3 Research Questions and Hypotheses

### 1.3.1 Research Questions

Research questions are the core part in the structure of the proposed research. They should state what the study would explore. The overall research question this thesis tries to answer is:

**How can security patterns be used to meet the security requirements elicited by risk analysis through model driven approach to evolve the security for web software applications?**

To answer the principal question, a series of detailed research questions are defined to address the problem as follows.

*RQ1: Why is there a need for a software evolution approach for security?*

*RQ2: How may the models be used to direct the whole process?*

- *How may the models be extracted from source code in the legacy systems?*
- *What kinds of models are required to reengineer the legacy systems?*
- *How may the models be used to reengineer the legacy systems?*

*RQ3: How may security requirements be elicited from the legacy systems?*

- *How may the risk be assessed for legacy systems?*
- *What kind of information is needed to represent security requirements?*

*RQ4: How may the elicited security requirements be satisfied by security patterns?*

- *How may the right security patterns be found to satisfy the elicited security requirement?*
- *How may the security patterns be integrated into legacy system models?*

*RQ5: How may the proposed approach be validated?*

### 1.3.2 Research Hypothesis

Based on the built research questions, a set of research hypotheses are presented. The underlying hypothesis of this thesis is:

**Software security engineering can be integrated during the software reengineering by using model driven approach, as a result, improves the security of legacy system.**

The principle proposition above is tested by program and model transformation and services implementation in the overall software evolution process. A subset of more detailed propositions can be derived as follows.

*RH1: Software design artefacts can be recovered as models from legacy systems. Models can represent the system artefacts thoroughly both at structure level and behaviour level.*

*RH2: Security artefacts can be detected from legacy system.*

*RH3: Security requirements can be elicited from the legacy systems.*

*RH4: Security requirements can be satisfied by adopting security patterns.*

*RH5: Security patterns are validated and can work properly to improve the software security.*

## 1.4 Original Contributions

In this thesis, a novel evolution approach is proposed with a set of frameworks, methods and models including a security evolution framework, a model slicing based reverse engineering method, a security requirement elicitation method, an ontology based security enhancement method as well as security domain models and design models. The original contributions of this thesis are summarised as follows:

C1: A novel software reengineering approach is created to integrate software security and software evolution for web application.

C2: A systematic comprehension method for legacy systems is proposed on the basis of model slicing and system partition. Different kinds of dependency relationships

among classes and objects in system diagrams are analysed based on which a dependency graph is constructed and sliced to facilitate the system understanding. The improved partition algorithm makes it possible to handle the analysis and decomposition for legacy system more effectively.

C3: An environment driven security requirement elicitation method for web application is proposed based on the proposed web applications classification method. Asset, threat and vulnerability are identified, quantified, treated as security vectors based on which security risks to the legacy systems can be calculated. Security requirements are elicited via the risk analysis and prioritised from the perspective of how urgency and importance of each one to the system security.

C4: A security enhancement approach is presented through integrating security patterns into the system design models. Security patterns are organised by the proposed multiple aspects metric. A security ontology for inter-relating elicited security requirements and security patterns is defined and realised in OWL. A security pattern search engine is designed to implement the selection of appropriate pattern for given security requirement by inferring the proposed security ontology.

## 1.5 Criteria for Success

A whole criterion for the success of the proposed method is how well they support successful software evolution for security demand. The following criteria are given to determine the success of the proposed research in this study:

- The proposed approach should be applicable in as many types of applications as possible as long as their source codes are available.
- The extracted models should be consistent to the original design and easy to understand.
- The security requirement elicitation framework should be able to reflect the system’s detailed level security requirement under the current environment.
- The proposed approach should be able to address the elicited security problems and provide implementation issues supporting for development.

- The generated models with enhanced security features should be reliable which makes it possible to perform forward engineering.
- The proposed approach should be feasible to realise. For example, it is possible to design and implement a real tool to demonstrate the approach.

## 1.6 Thesis Outline

The rest of this thesis is organised as follows:

Chapter 2 provides a general overview of research background and basic concepts related to security driven software evolution. Related works are discussed from three main aspects, software evolution techniques, software security techniques and model driven security.

Chapter 3 introduces a unified approach, SEMDA, for security driven software evolution using model driven approach. The architecture and processes of SEMDA are proposed.

Chapter 4 describes the approach to legacy system understanding and extraction. A model slicing method is proposed on the basis of analysing different dependence relationship among classes and objects in UML models, which serve as the foundation of further system decomposition. During the understanding, the existing security mechanisms are detected using the proposed two phase method to help the decision making whether or not the legacy system is security enough to protect the user's security objectives.

Chapter 5 describes an environment driven risk assessment method using security vector to elicit the security requirements. A web application classification method is proposed to help the threat analysis.

Chapter 6 describes the advantage to use security pattern as security improvement method to satisfy the elicited security requirement stated in Chapter 5, as well as a security ontology is developed to associate the security requirements with the corresponding security patterns.

Chapter 7 describes how the experiments are performed on a case study, which

demonstrates that the proposed approach works in practice.

Chapter 8 draws a conclusion about the proposed approach as well as outlines the limitation. The research questions are revisited and answered. The future work is discussed as well.

Appendix A lists security patterns repository classified by the proposed classification approach.

Appendix B lists source code written in OWL to develop security ontology.

Appendix C lists all the related publications by the author during the PhD study.

# Chapter 2

# Background and Related Work

**Objectives**

- To introduce software security and its related concepts.
- To present an overview of software evolution.
- To present an overview of Model Driven Engineering
- To clarifies basic concepts related to security driven software evolution

## 2.1 Overview

Unlike the following related studies, this research will integrate security engineering with software evolution. A systematic approach is proposed to support the software evolution focusing on security concerns. The techniques, such as model extraction, model slicing and system decomposition for legacy system understanding, risk analysis based security requirement elicitation, security pattern used to apply expert security knowledge in the extracted models to improve the security level of legacy system, are used in this research to complete the functions of the proposed general framework.

For a better understanding of the proposed framework, background and related techniques used in this research are reviewed in the following section.

## 2.2 Software Security Engineering

Software almost controls everything in your life. This is aggravated with the rapid development of Internet. More and more network-based software have been developed which greatly simplify data accessibility. However, software provides the diverse

functionalities as well as introduces the security issues [111]. More and more security vulnerabilities are emerging and exploited by insider or outsider of the organisation which leads to financial losses and bad impact of organisation's credibility as well.

Security as an important non-functional characteristic of software system has been increasingly thought as an essential part of the software development lifecycle. Mouratidis et al. [128] takes the first step towards this idea and proposes that security should be integrated into the software engineering.

In the paper [110], Gary McGraw breaks down software security problem into two sub-problems: software security problem and application security problem. Software security, on one hand, is the process of designing, implementing, and testing software for security, identifies and solves security problems in the software itself. Application security, on the other hand, is about protecting software to operate in a specific way after completing the development, finding and fixing known security problems.

In this section, an introduction to the field of software security is presented. This introduction includes a general overview, followed by taxonomy of software security areas.

### 2.2.1 Software Security Concept

The concept of software security has been considered as relevance to threats, attacks and vulnerabilities. Several key terms associated with software security are given as follows. According to the Department of Homeland Security, National Cyber Security Division [59, 168]:

**Secure software** *is software that is resistant to intentional attack as well as unintentional failures, defects, and accidents.*

**Software security** *is the ability of software to resist, tolerate, and recover from events that intentionally threaten its dependability with the preservation of availability, integrity, and confidentiality of information.*

**Security engineering** *is about building systems to remain dependable in the face of malice, error, or mischance. As a discipline, it focuses on the tools, processes, and methods needed to design, implement, and test complete systems, and to adapt existing*

*systems as their environments evolve* [6].

In order to make a precise description of the proposed model, the involved concepts from [111] are listed as follows:

**Asset** *is "anything that has value to the organisation" and which therefore requires protection.*

**Stakeholder** *is an organisation or person who places a particular value on assets.*

**Security objective** *is a statement of intent to counter threats and satisfy identified security needs.*

**Threat** *is the potential cause of an unwanted event (i.e., an attack), which may result in harm of a system or organisation.*

**Attack** *is an attempt to destroy, expose, alter, disable, steal or gain unauthorised access to or make unauthorised use of an asset.*

**Attacker** *is an entity that carries out attacks.*

**Vulnerability** *is a weakness of an asset or control (i.e., in ISO 27000-series a control is a synonym of a countermeasure), which may be exploited by a threat. This general definition covers all threats categories.*

**Countermeasure** *is an action taken to protect an asset against threats and attacks.*

**Risk** *is the combination of the probability of an event and its consequence.*

### 2.2.2 Security Properties

Information is a very important ingredient in software and its security can be achieved by three globally accepted properties CIA (Confidentiality, Integrity, and Availability) [10].

**Confidentiality** *is prevention of unauthorised disclosure of information.*

The main objective of confidentiality is to guarantee that information can only be accessed by authorised user, no matter how it is accessed and where it is kept. Mechanisms like encryption, password, access control, biometrics, privacy and ethics can be used to maintain confidentiality.

**Integrity** *is prevention of unauthorised modification of information.*

The main objective of integrity is to ensure the accuracy and consistency of information from being tampered intentionally, unintentionally, or accidentally. Mechanisms like auditing and configuration management can maintain the integrity and guarantee reliability and privacy of information.

**Availability** *is prevention of unauthorised withholding of information.*

The main objective of availability is to guarantee the authorised users can access the information and service whenever needed. Mechanisms like disaster recovery plan, resumption plan and data backup plan can be used to ensure the availability.

Confidentiality, integrity and availability (CIA) are commonly used as security features of information system. Other security features are accountability, authentication, authorisation and non-repudiation.

## 2.3 Legacy System and Software Evolution

Software systems need continuous evolvement, in order to deal with constantly changing software requirements. As a process of conducting continuous software reengineering, software evolution is repeated software reengineering [183]. Software reengineering technology involves reverse engineering and forward engineering, and has becoming a practical solution to the problem in legacy system evolution.

Legacy systems pose many conventional challenges to software maintainers. Nevertheless, in order to reduce cost of software development, organisations have to maximise the benefits from legacy assets (software system). The increasing cost of managing legacy systems together with the need to preserve business knowledge has meant that renovating legacy systems has become an important research topic over the years. Thus, maintaining functionalities and keeping up with changing business or technical conditions are considered as two important and urgent tasks.

### 2.3.1 Legacy System

As legacy software systems no longer meet the needs from customer’s requirements,

emerging operating software and hardware environments, they are subject to evolve. The maintenance scope has to cover not only maintenance of the existing functions, but also modifications to the current architecture and functions so that adding requirements will be fulfilled. In addition to such changes, non-functional changes especially when security as a requirement is concerned in this thesis may also be a consideration typically when software system entails adaption of a new computing environment.

Bennet defines legacy systems informally as “large software systems that we don’t know how to cope with but that are vital to our organisation [14]”, while Brodie defined it as “any information system that significantly resists modification and evolution [18]”. Whichever, a legacy system is the one that is still valuable, but is difficult to maintain.

A significant number of legacy systems remain in operation because they are critical to the business processes which they support. As Warren stated “the combination of extended lifetimes and poor maintainability means that legacy systems are expensive to change and in many cases, they cannot accommodate emerging requirements. This is clearly an undesirable situation which, until recently, has been tackled by replacing the system or attempting to maintain it [173].”

On one hand, it is dangerous to replace a legacy system since there may exist a risk that vital business knowledge may be lost since it is embedded in the old systems very well. On the other hand, legacy system is difficult to maintain and expensive to change.

Reengineering as the process of software evolution is a relative new approach to solve the two extremes of system replacement and continued maintenance by improving the system in some way and results in a system which is more responsive to change.

Compared with replacement, the costs of reengineering a system are typically lower. Reengineering can also reduce the major risks resulted from replacement and continued maintenance. Reengineering starts from the current legacy system so as to the risk of losing vital business knowledge built in it can be reduced. Different from maintenance, the result of reengineering is an evolved system which can meet the new requirements in a cost effective manner [173].

### 2.3.2 Software Evolution and Reengineering

Software evolution is the process of modifying an existing software system to satisfy an enhanced set of requirements. Software reengineering is a systematic way to perform software evolution. In particular, Chikofsky and Cross define reengineering to be “the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form [27]”. Altering existing systems involves the majority of all software development time and expense, and evolution comprises the majority of system alteration (maintenance) activities.

When evolving a legacy system, the artefacts of the system have to be taken into consideration which is different from developing a new system. The major challenges involve ensuring the new requirements are consistent with those of the existing system, maintaining system’s architecture, code review and understanding of the current system, and maintaining the design integrity in conceptual level. The legacy systems can be migrated to a new platform, language, hardware, operating system, or to a new software development paradigm, or integrating with other systems with the help of reengineering technology.

Software evolution comprises continuous reengineering process. The difference between them lies in that reengineering is a single change cycle, while evolution is a broader concept which implies the repeated reengineering activities and maintenance activities. Software reengineering technology is the practical solution to the problem of evolving legacy system to meet the ever-changing system requirements. There is a trend that the needs and costs of changing software are increasing with the rapid development of computer software and hardware.

Software reengineering is an integration of reverse engineering and forward engineering. Bachman [9] introduced a chart of software reengineering cycle for better understanding the process of software reengineering, shown as Figure 2-1.

Reverse engineering is the process of comprehending a target system by identifying the components and their inter-relationships within the system and creating representations of the system in another higher level of abstraction form.

Forward engineering is the process like traditional software development that proceeds

from high level abstraction, design to the implementation. It aims to improve a software system by taking consideration of new functional and non-functional requirements for the migrant system. Forward engineering consists of a sequence of activities, including new requirements elicitation, system transformation, and finally system deployment in the new environment.

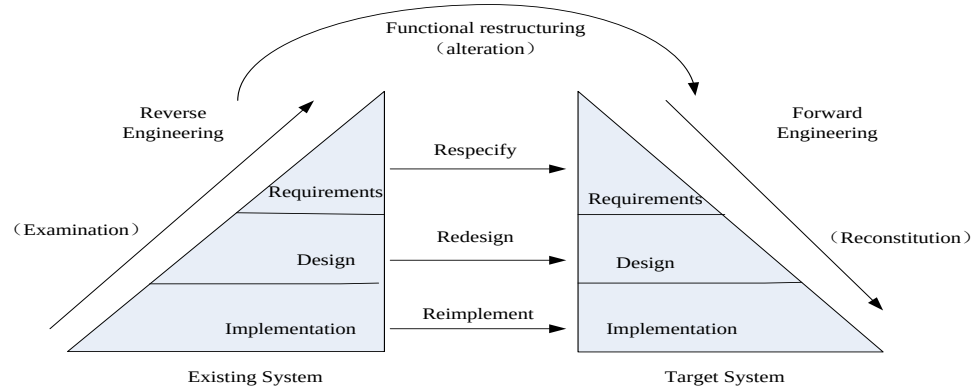



Figure 2-1 Software Reengineering Process [9]

The following research areas fall into reverse engineering domain:

- Reengineering requirement aims to clearly identify the reengineering objectives in the legacy system to be migrated.
- System analysis involves source code analysis and abstraction models to evaluate the legacy systems from the functional, technical and architectural aspects points of view.
- Positioning involves improving the qualities of the legacy system by restructuring without changing the external behaviour.
- Re-documentation is the process of changing the legacy system in a different view while keeping it in a semantically equivalent representation for better understanding.
- Design recovery focuses on identifying the meaningful abstraction of the system in a higher level and associating code with specific function.

## 2.4 Software Security in Software Reengineering

Traditionally, security is viewed as an organisational and Information Technology (IT) systems level function comprising of routers, firewalls, intrusion detection systems (IDS), system security settings and patches to the operating system (OS) and applications running on it. Until 2001, the first book [168] on the software security topic appeared, demonstrating how recently developers, architects, and computer scientists have started systematically studying how to build secure software. In this book, Viega and McGraw point out that software vulnerabilities and software exploits are the root cause of computer security problems. In the next year, an article "Why Is Software So Bad?" points out that bad habits and inadequate software life cycle processes have led to the developments in software engineering and the development of poor software [57]. In this section, a series of software reengineering efforts in the security domain are reviewed.

### 2.4.1 Security Engineering Process

There are several processes for secure software development in the field. Based on a survey on security engineering processes [32], three of the best known processes are presented, which are OWASP's CLASP [138], Microsoft's SDL [79] and McGraw's Touchpoints [111]. All of them provide an extensive set of activities covering a broad spectrum of the development life cycle.

#### 2.4.1.1 CLASP

CLASP [138] is a lightweight process for building secure software, which is contributed and reviewed by several leading security companies of the OWASP. Twenty-four top-level activities are included in the CLASP and can be tailored according to the development process. Key features include:

- The main goal of CLASP is to support the secure software development.
- CLASP involves a group of independent activities which can be integrated in the development process.
- Two road maps: legacy and green-field have been defined to give some guidance

on how to combine the activities into the ordered set.

- It defines some roles that may have an impact on the security and assign the tasks to the according roles.
- CLASP is open to public and rich resources are provided to facilitate the implementation of the activities.

#### 2.4.1.2 SDL

Microsoft defined the SDL [79] to address the security issues they frequently faced in their products in 2002 which is composed of a set of activities aiming at addressing security issues in their development process. SDL can be characterised as follows:

- SDL is designed as an add-in to the software construction process.
- The SDL process is well organised and related activities are grouped in stages, it is direct to map the SDL activities to the standard software development process.
- SDL specifies the method in a concrete and practical way.
- SDL provides rich resources in documents and supporting tools in MSDN.

#### 2.4.1.3 Touchpoints

McGraw emphasised that software security can be implemented by using engineering methods and proposed Touchpoints [111] as a set of best practices distilled over the years out of his extensive industrial experience. Seven so-called touch points are grouped of the proposed best practices, which are: principle, guideline, rule, vulnerability, exploit, attack pattern and historical risk. Touchpoints can be characterised as follows:

- Risk assessment is of importance in Touchpoints when it comes to software security.
- Black-hat and white-hat activities are provided as Black-hat activities are about attacks, exploits and breaking software, while White-hat activities are more constructive in nature and cover design, controls and functionality.
- Touchpoints is flexible and can be tailored to existing software development

process.

- Touchpoints is rich in examples and resources.

#### 2.4.1.4 Process Support for Change

The general characteristics of the three processes described above are summarised in Table 2-1 with the discussion how the processes dealing with evolution.

Table 2-1 Comparison of CLASP, SDL and Touchpoints [32]

| | CLASP | SDL | Touchpoints |
|---|---|---|---|
| General | | | |
| Focus | Security at centre stage | Security as supporting quality | Risk management |
| Structure | Independent activities | Well-organised set of activities | Best practices |
| Guidance | Rich set of resources | Concrete activities | Rich examples and resources |
| Evolution | | | |
| New security vulnerability | Software updates Security advisories | Software updates Security advisories | Not Supports |
| Change in security assumptions | Not explicitly supported | Some continuous activities | Not explicitly supported |

All of the security engineering processes mentioned above provide a set of activities covered the whole software development life cycle. However, there is no explicit support for evolution in these processes [32].

### 2.4.2 Security in Requirement Engineering

A security requirement is a manifestation of a high-level organisational policy into the detailed requirements of a specific system. Security policies are complementary to the functional requirements of a system, which are a kind of non-functional requirement.

In recent years, there is recognition in the researches of software evolution that changes in stakeholders' needs (expressed as requirements) has become as one of the main drivers of software evolution [130]. Requirement engineering researches focus on addressing the abilities that the system can provide. Therefore, lots of attention in this area is given to the functionality specifying what the system should do. Little attention is given to the requirements of what the system should not do. The consequence is that the resulting system can't be effectively assessed prior to implementation without the properly defined security requirements.

In this section, approaches and related work of security requirements are reviewed. The approaches are classified according to their different concentrations, namely: secure requirement process, security requirement elicitation, security requirement analysis and from security requirement to security architecture.

#### 2.4.2.1 Secure Software Requirements Process

Secure requirement process emphasises on the steps for analysing security requirements. The steps may comprise risk analysis for identifying security vulnerabilities and exploration of countermeasures for addressing identified weaknesses. In this section, three typical secure requirement approaches have been examined.

**SREP:** Mellado proposes the SREP [117], the abbreviation of Security Requirements Engineering Process, which provides an approach to deal with reusing security requirement systematically. In order to achieve this, a security resource repository is proposed and Common Criteria are integrated into the software development lifecycle.

**Security Requirements and Trust Assumptions [75]**: Hatebur et al. propose a security requirement engineering process to develop security systems based on problem frames, and a collection of security patterns, plus components as the way to cope with the solution. The components in their approach cover the roles and security goals statements, relationships of security requirements with other system requirements, threats and validation the elicited security requirements.

**SQUARE [113]**: The Security Quality Requirement Engineering (SQUARE) is a comprehensive method to elicit, analyse, categorise, prioritise, and document security

requirements for software systems. The method aims to elicit security requirements as a part of requirement engineering process rather than an afterthought. The main activities in this method involve identification of security goals, risk analysis for identification of threat to security goals and elicitation of security requirements with which security goals can be accomplished by satisfying them. The method adopts “waterfall model”, and it doesn’t provide iterations. Therefore, elicited security requirements can’t be revised and evolution of a system can’t be supported.

#### 2.4.2.2 Security Requirement Elicitation

Security requirement elicitation focuses on the process or the approach to identify security requirements. Various security requirement elicitation techniques are proposed in the security requirement domain. There are number of proposals for eliciting security requirements using techniques such as abuse case [109], misuse case [4, 111, 154], security use case [50], common criteria [172] or attack trees [40].

**Abuse cases** [109]: Abuse case is used to specify the interaction between the users and the system, where harm can be caused to the resource owned by the user, stakeholders of the system during the iteration. An abuse case should illustrate the abuse use of privilege, which makes a further distinction. The strategy using to describe use case can be used to describe the abuse case. Obviously, abuse can be performed by modifying system software to gain total control of the target system. Security requirements can be elicited by illustrating the scenario of abuse case.

**Misuse cases**: Guttorm Sindre and Andreas Opdahl [154] extend use case diagrams with misuse cases to represent the actions that the systems should prevent against. Misuse case is the use case with negative representation from the view of attacker by personifying the threats threaten to the system. A misuser is the user who is the inverses of a legitimate actor in the use case diagrams. The threat can be used to elicit security requirement which is then satisfied by providing additional functionality in the new use cases or modifying existing ones. Ian Alexander advocates using misuse cases and use cases together to conduct threat modelling during requirement analysis [4]. Misuse cases elicitation is a relatively recent approach to address and analyse security threats.

**Security use cases**: Similar approach is proposed by Firesmith in [50] named security

use cases. Security use cases have some similarities with misuse case because both methods focus on depicting the idea of negative scenarios. Security use cases can be used to specify the security requirement related to how a system should protect itself against the relevant security threats.

**Common criteria with use case** [172]: This approach elaborates how common criteria can be integrated with use case diagrams. The aim of this correlation of common criteria with use case is to deal with security issues in IT products during the software development life cycle. The profile of each actor involved in the use case diagram can be completed in a mandatory manner with relating common criteria with use case. There are seven fields in the actor profile which are name, type, location, use case association and whether or not private or sensitive information are involved in the use case. Actor profile can be used to map threats from predefined threat classifications to the actor after the actor profile is completed by the use case creator.

**Attack tree**: Attacks trees [40] are a representation of attacks by using trees as the data structure. The root node of attack tree is the attack's goal while the leaf nodes are the different ways to achieve the goal. The attributes associated with each node can be used to analyse the value of attack tree.

Besides, based on Sindre's misuse cases, Firesmith's security cases and operational model of computer security, Kassem Saleh and Maryam Habil [147] introduce a comprehensive framework named Security Requirement Behaviour Model (SRBM) for dealing with security requirements for web services and web applications. A template for eliciting misuse cases is provided in SRBM.

#### 2.4.2.3 Security Requirement Analysis

Security requirement analysis emphasises on how to model, specify and analyse security requirements. Several typical methods are reviewed as follows.

**SecTropos** [126]: As a methodology of software development, Tropos is based on modelling framework i* [103] by using which the system and its organisational environment can be modelled. There are three main concepts in the Tropos models, which are actor, intention or goal, and dependency among them. Security concerns can

be modelled in the software development life cycle by extending Tropos abbreviated as SecTropos. Security constraints, trust among security entities and permission delegation are modelled explicitly in SecTropos. SecTropos provides a systematic method to elicit and analyse security requirements, however, it can't make changes propagated between the different models [130].

**Secure i* Framework [102]:** i* is an agent oriented requirement modelling framework. Liu et al. [102] extends i* with the ability to analyse security and privacy requirements. System stakeholder, potential attackers, and agents are three actors in their approach. By studying the relationships between the three actors, security requirements can be elicited and analysed by Secure i*. The techniques implemented in the approach are dependency vulnerability, access control, analysing attackers and countermeasures. Some limitations of the approach have been described in [130] as there is no guarantee identification of all potential attackers and countermeasures are likely incomplete.

**KAOS** [163]: Van Lamsweede [164] proposes an approach to modelling, specifying, and analysing security requirements by extending their earlier framework [165] to elicit goals and identify anti-goals to security. Anti-goals are security obstacles and are similar to the idea of misuse case. KAOS is a goal-oriented methodology for building requirements and eliciting requirements from KAOS models in the requirement engineering. In KOAS, a requirement is treated as a realisable goal under responsibility of an agent in the software-to-be. Building on KAOS, Landtsheer et al. propose an approach [31] to check requirements models for violation of confidentiality properties.

**UMLsec [83]**: Jurjens proposes an approach to extend the UML with the ability to modelling security concerns. With UMLsec, application developer can model the security related functionality in the system design phase and analyse security on the system model to verify whether or not it satisfies the security requirements. UMLsec assumes that requirements have already been identified and there exists some system design to satisfy them. Several systems such as mobile communications [86], automotive [15], and banking [84] have been applied to validate the application of UMLsec.

**SecureUML [104]**: Similar to UMLsec, Lodderstedt et al. propose a modelling

language named SecureUML in the software design phase. Based on UML, SecureUML focuses on modelling authorisation policies and how these policies can be integrated into model driven software development process. Role-based Access Control is used as the meta-model to specify and enforce security. Different from most of security requirement analysis methods, SecureUML doesn't provide any analysis on eliciting security threats, but focuses on authorisation constraints.

#### 2.4.2.4 From Security Requirement to Security Architecture

As two products in software development life cycle, requirements and architecture are closely coupled with each other. Software architecture plays a key role in non-functional requirement especially security, evolvability, software reliability and so on. In this section, two approaches to moving security requirements to security architecture are reviewed.

**From System Goals to Software Architecture [163]**: Based on the KAOS [165] framework, Van Lamsweerde proposed a goal-oriented approach to architecture design. Security requirements are modelled, specified and analysed by the means of KAOS and then transformed into architecture step by step. The proposed approach is based on refinement and it may be insufficient when there are propagated needs such as bottom-up or middleware deployment.

**Transforming Security Requirements into Architecture [186]**: Yskout et al. introduced a semi-automated transformation for some key security requirements, namely authorisation, auditing and delegation. The approach starts from defining specific meta-model to ease the transformation from security to architecture. However, the result in that approach is not mature and a number of aspects need to be further elaborated [130].

In this section, related work on security requirement are reviewed and organised by separating different concerns. It is no doubt that security requirements play a key role in software security engineering. The exiting research on security requirements provide complex and heavyweight approaches to requirement analysis and management ranging from model-based to goal-oriented. However, there is little work on security requirements dedicated for web applications. In this research, a lightweight approach for

web application is proposed to facilitate the security requirement elicitation for security novice.

### 2.4.3 Software Security in Software Design

Security of information system consists in identifying the vulnerabilities, evaluating the threats, determining the risk which vulnerability allows threat given to be carried out, and therefore, it uses methods, techniques and tools to protect the resources of information system in order to ensure the availability of the services, the confidentiality and the integrity of information.

- The availability of the services: the services and information must be accessible to the authorised entity when they need some.
- Confidentiality of information: information does not belong to everyone and it can only be accessed by those who have the right of it.
- Integrity of information: information (files, messages…) can be modified only by the authorised entity.

Adding security solutions to a system that has already been functionally realised is very difficult, and can make the system instable. The security requirements should then be integrated at the design stage, so that they can be identified with the first parts of development process. The posteriori security of critical systems (firewall, antivirus, etc.) does not constitute the best security policy. The development of a security policy must be done at the same time in the functional design stage, and the final model must integrate the functional along with security specifications.

In this section, the review of security implementation in design phase is carried out from the following aspects. Firstly, security architecture is introduced as the foundation for the following sections. After that, the security design approaches are classified according to what they are implemented as, in a more technical term is the “granularity” they are founded on, namely: components-based, aspect-oriented and service-oriented.

#### 2.4.3.1 Software Security Architecture

As Barais said in [11] “a software architecture describes the structure and behaviour of a

software system. In a software architecture specification, a system is represented as a set of software components, their connections, and their behavioural interactions". Software design process will benefit from the creation of software architecture as it promotes comprehension of the system. Moreover, it provides the basis of precisely analysing software design which makes early detection of design flows and errors possible and thereby leads to improvement of software quality and facilitates to ensure correctness. Architecture plays an important role in non-functional requirements research including security requirement.

Security design at architectural level is critical to achieve high assurance software systems. A secure architecture describes how security requirements are enforced in a software design and it is a high-level design representing all the components, connectors and how they are organised to meet the security requirements. Specially, a secure architecture depicts how security countermeasures are deployed among the design artefacts to achieve the security features, such as confidentiality, integrity and availability.

The question on how to create an architecture that has certain security qualities needs to be answered. Often, security patterns [65, 186] are used to this aim. The non-functional requirement (NFR) framework also uses patterns to create secure design [60, 178]. In [163], Van Lamsweerde proposes a patterns-based approach to creating architectures. A detailed security pattern review is given in Section 2.8.

Besides patterns, security principles are also commonly used as the guidance for creating secure architectures. For instance, in [108] an attack surface metric is proposed, which can be used to measure the security of a design and improve it. Equally, the principle of least privilege can be used as the guidance for improving the security of architectures [22].

#### 2.4.3.2 Component-based Security Approaches

Component-based software engineering represents the concepts of assembly and coupling of components-essential to most engineering disciplines [87]. Component-based approaches in security architecture evolution focus on improving or enhancing security in software architecture level by implementing security as

components or connectors to achieve the goal. Many approaches have been proposed for the security issues of component-based system. In this section, several classic methodologies are selected to review because the majority of them try to solve the security problem from the evolution point of view.

Shin and Gomaa [153] propose an approach to modelling the process of evolving non-secure applications into secure applications in the light of modelling the software requirements and software architecture built on distributed environment. Security use cases have been used to achieve security requirements in their research. They evolve the non-secure applications into secure application by encapsulating the security services including integrity, confidentiality, non-repudiation and access control into the corresponding connectors which can be invoked if the security requirement conditions hold. Although their research presents an overall method to integrate security into the application system, only architecture level evolution is illustrated in detail, and several security services are addressed in a general way.

Cotroneo et al. [29] present an approach to improve the security level of an existing system by separating the concerns and reusing them in a multi-layer architecture. Security is expressed in terms of confidentiality, integrity and availability. The system relies solely on COTS technologies. The overall system architecture is composed of the clients, the middle-tier server, and the group of replicated legacy application instances. The middle-tier adds security to the legacy application, which runs in the back-end. Business middle-tier is used to handle security mechanisms. Confidentiality is achieved via cryptography while integrity and availability are obtained through replicating the functional modules.

Ren et al. [143] propose a Connector-Centric approach which argues for a comprehensive treatment of security on architecture level based on software connectors. Connectors play the suitable role in modelling, capturing and enforcing security in that research. The approach is clarified by a classic access control model using the following core concepts: principal, privilege and context. Connectors play a key role in this approach. It contributes more comprehensive treatment of architectural security.

Gasmi et al. [55] propose Security Meta-model of Software Architecture (SMSA), a

software architecture meta-model that takes consideration of the security concept separately from functional components by means of secure connectors, so as to contribute a more complete and deeper modelling of software architecture. It is implemented by integrating the security concept as a non-functional requirement and facilitating the detection of points that request the security mechanisms implementation during the exchange of information and the communication of the various distributed application elements.

#### 2.4.3.3 Aspect-oriented Security Approaches

Increasing attentions have turned to addressing security concerns using Aspect-Oriented Software Development (AOSD) approach because of crosscutting characteristics of security in nature.

Aspect-Oriented Software Development (AOSD) [88] aims at separating the concerns in software development and it is often used to address the complexity when new concerns need to be integrated into the architecture.

As described in [33] “the central problem in modifying the security aspects of a legacy system is the difficulty of identifying the code that is relevant to security, changing it, and integrating the changes back into the system”. Hence, with the ability to separate concerns, Aspect-Oriented Software Development (AOSD) is a good choice to deal with security evolution only if security is treated as one of the system’s concerns. Here, some excellent works are reviewed to show how security is implemented with AOSD.

Laney et al. [97] propose an approach to evolve the system with the support of employing aspects. A legacy C/S application is selected as case study with “message tampering” as attack. The approach shows how aspects are used to support the legacy system evolution for mitigating the specific security attack. As the security countermeasure to message tampering, digital signature is implemented as an aspect to improve the security.

Georg et al. [56] present a methodology to incorporate security countermeasures into an application by using aspect-oriented modelling (AOM). The main idea of their approach is to model the security mechanisms and security attacks as aspects, which involves four

steps as follows. Firstly, risk analysis is used in their approach to analyse the system in order to identify the potential threats to the system assets and model the identified threats as attack aspects. Secondly, misuse models are generated by combining the application's base model with the attack aspects. Thirdly, an evaluation is made to measure the impact of an attack by analysing the misuse models. Finally, if the evaluation result shows that an attack may pose severe impacts to the system, alternative solution is analysed to identify the security mechanisms or countermeasures to cope with that attack.

Mourad et al. [125] present a method to harden security concerns by enabling the application with additional security requirements based on Aspect-Oriented Programming (AOP). Pointcuts and primitives are proposed to AOP language for realising security concerns. Two pointcuts are proposed to identify particular join points in a program's control-flow graph (CFG) while two primitives are used to pass parameters between the pointcuts. The program's call graph is analysed to determine how to change function signatures for passing the parameters associated with a given security hardening. Their approach ends with the algorithm implementation and case studies explanation.

Zhu et al. [189] propose an intrusion-aware framework to build secure software based on Aspect-Oriented Software Development (AOSD). Security concerns are treated as crosscutting concerns and modularized using aspects. Security artefacts, such as attack scenarios, intrusion detections, are modelled as aspects by using an aspect oriented (UML) profile. The intrusion detection aspects are implemented and woven into the target system based on the UML models. An experimental evaluation by applying this framework for some of the most common attacks is presented in their work.

Welch et al. [179] propose an approach to reengineering a third-party application by designing a reflective security architecture which aims at reducing the tangling between application code and security code. The application is designed using proxy pattern and inheritance pattern to improve the security. Security is treated as crosscutting concerns and the separation between them is achieved by using aspect composition and analysis techniques. A similar approach is described in Xu et al. [182].

#### 2.4.3.4 Service-oriented Security Approach

To cope with the increasingly complicated requirements of software systems, enterprises are likely to adopt service-oriented architecture (SOA) to align their systems with their business processes, using web service technologies as best practice approach to establish an SOA. Security as the crosscutting concerns are hard to integrate into the overall SOA development process, while web services are best suited to implement key concerns. Considering the amount of overly complex web service security standards, security is often an afterthought which resulting in independent silos of security infrastructure. A solution to this problem is to provide security as a service by using service-oriented approach, for example offering a collection of services, providing the central functionality of security services such as authentication, authorisation and policy management. These services form a security architecture [34, 42].

In this section, methodologies of implementing security as a service are reviewed. As Buecker et al. stated [20], "a service is representative of a repeatable business task. Services are used to encapsulate the functional units of an application by providing an interface that is well defined and implementation independent".

Hafner et al. [68] present a reference security architecture called SeAAS which transforms the secure software as the security services and thereby implements security as a service. An illustration is given in their approach by using SeAAS to solve non-repudiation requirement.

Han et al. [73] introduce a service oriented framework to compose and evolve security concerns. It allows the system developers to design required security concerns into services. Security properties are specified using a semantic model and security services are composed and evolved using negotiation and re-negotiation techniques. Security compatibilities between the realised security services and the system's security goals are checked using analysis techniques.

Emig et al. [41, 42] propose an interface to provide security services to web services and service-oriented applications. The same to other security service oriented methods, their approach treats security as crosscutting concerns and security is provided by services. Security services, such as authentication and access control, are provided by

corresponding interfaces in the security architecture described in [42] and [41]. An administration interface is designed to manage the access control policies, users and groups.

Yamany et al. [39] propose a security framework based on SOA to provide two important security services: authentication and authorisation. The security services are said to be intelligent, automatic and reusable because mining techniques have been used in their design in which clustering mining algorithms are designed to represent and automate the access control rights, association rules are built to facilitate the prediction of attacks. Moreover, they explore their method in a case study to demonstrate the behaviour of the proposed security services in SOA environment.

In this section, existing research approaches demonstrate how security features are implemented in architecture and design stage of software development lifecycle. Security levels are improved via implementing security features as connectors, aspects or services. Though several approaches focus on the security enhancement for existing applications such as the work in [29, 97, 153], none of them provides the comprehension of target application and thereby generating security requirements to specify which assets need to be protected against which threats to which security extent. In this research, a comprehensive approach to security evolving for existing applications is proposed from software reengineering perspective involving security analysis from the understanding of target system, eliciting the security requirements and reconstituting the system to satisfy the identified security requirements.

## 2.5 Model Driven Engineering

The term “model” is used in many contexts and often has different meanings. As described in [25], “a model can mean an abstraction and representation of the important factors of a complex reality, which is different from the thing it models, yet has the same main characteristics as the original”. Model is a direct representation of the answer to a given problem in simpler context and thereby it is easy for people to understand in a natural way.

A model can help people work at higher level of abstraction by hiding the details and

generating big picture. With complex mechanical or electronic machines or large buildings, the designers and constructors have some models (e.g. blueprints and floor plans) which provide an accurate overview and geometric representation of the structure. In software development domain, a model plays the similar role by creating a model of the system for better understanding.

Modelling is the process to bring out models which is an essential part of developing large complex software systems. It is common for an experienced software developer to take more time to build models than to write actual codes. Models have the advantages over source code because (1) they can convey information in a more efficient way; (2) they can enhance the understanding of the system; (3) they provide an easy manner for people to share knowledge; (4) well-constructed models make it easier to deliver complex and large systems on time and within budget.

Model driven engineering is the systematic way to use models to guide software development. In this section, a browse of model driven related concepts and research are reviewed to lay a better understanding of the proposed framework in this thesis.

### 2.5.1 Model Driven Architecture

Model Driven Architecture (MDA) is an approach to application design and implementation which is produced by the Object Management Group (OMG). The core idea of MDA is to use models and modelling as the main artefacts and activities in software development, which increases the power of models. The term “model-driven” means that it provides a way for using models to direct the process software development involving understanding, design, construction, deployment, operation, maintenance and modification [135]. The term “architecture” means a specification of the components and connectors of the system and the rules for the interactions of the components using the connectors [135].

Just as the name implies, models play a core role in MDA based development. There are different levels of models in MDA, involving Platform Specific Model (PSM), Platform Independent Model (PIM) and Computational Independent Model (CIM). A PSM is model containing all required information regarding to a specific platform which can be used by software developer to implement the executable code. A PIM is the model

describes the structure and behaviour of the application regardless of the implementation platform. A CIM is the most abstract one in MDA which represents the context and purpose of the model without any computational complexities.

A distinction of models is suggested by Fowler in [52]. Three levels of models are proposed, namely Conceptual Models, Specification Models and Implementation Models. The same to CIM, conceptual Models are more abstract and used to describe concepts. Specification Models are more specific and used to describe the system to be built without specifying the implementation details. Different from PSMs, Implementation Models specify how the system has to be implemented. Fowler emphasises that each type of models can be illustrated using same modelling language [52].

Mellor and his colleagues in [118] classify models as sketch model, blueprint model, or executable model. A sketchy model is not complete or precise, being used as specification to describe an idea or description to ease of understanding and simplify communication. A blueprint model is more precise and complete which can be used as specification to build a system. Executable Models, such as Executable UML, can be directly interpreted by a processor or to generate an executable system.

The concept of Model Driven Development (MDD) is come out as a generalisation of the MDA approach for software development. In [119], MDD is defined as the notion that we can construct a model of a system that we can then transform into the real thing.

A common pattern of MDD is to define a PIM, and to apply transformations to this PIM to obtain one or more PSMs and then generate the code based on the PSMs. Figure 2-2 shows the sequence by using MDD.

This MDD development approach proposes not only a collection of models representing the system at various levels of abstraction, but also a course of software development [120]. The main benefit of MDD stems from the automatic transformation process that may reduce the costs of software development and enhance the quality of the developed software as well. In order to support automation, machine readable models are required to be automatically transformed using tools into different development artefacts, including schemas, code skeletons, test harnesses, integration code, and deployment

scripts for various platforms.

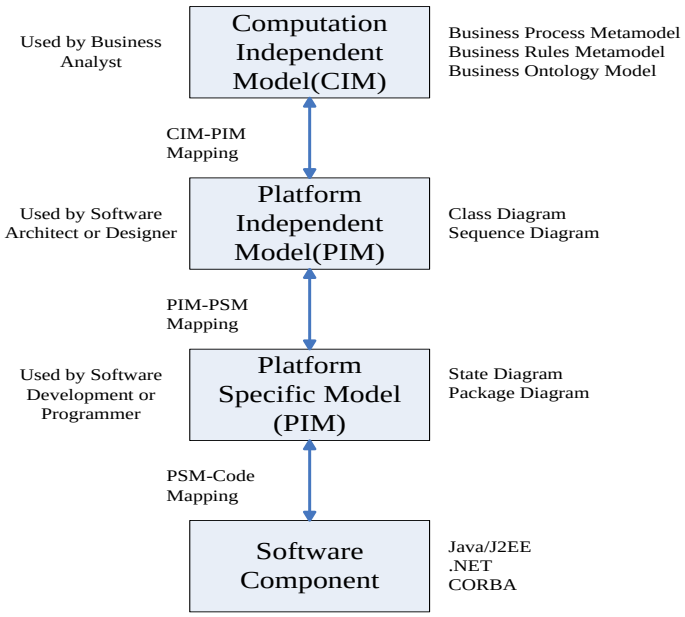



Figure 2-2 MDA Development Sequence [74]

To enable automatic transformation of a model, the model written in a language must obey the following definitions [93]:

> **A model** *is a description of (part of) a system written in a well-defined language.*
>
> **A well-defined language** *is a language with formal form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer.*

Modelling languages define what models are considered to be valid. They describe the allowed model primitives and the rules specifying how these primitives may be combined to form a valid model. The rich the primitives, the more different aspects of the problem domain are represented by using provided primitives.

MDA provides a framework based on the Unified Modelling Language (UML) and other industry standards for visualising, storing, and exchanging software designs and models, which include the Unified Modelling Language (UML) [134], Meta-Object Facility (MOF) [135] and XML Meta-Data Interchange (XMI) [133], etc. In the context of MDA, much effort has been invested in MOF, language definition and extension mechanisms (UML and UML profiles), model transformation specification (MOF Query/View/Transformation RFP [136]), and tool support. These developments

constitute enabling technologies to model-driven development.

### 2.5.2 Model Driven Reengineering

Models are useful to specify how to build a system or to describe how an existing system is built which lead to two different kinds of models, specification models and descriptive models. Specification models are generated and used to specify the new development systems while descriptive models are derived from existing system. Models are common foundations of MDE research and reverse engineering research which result in lots of share in both disciplines [43]. Therefore, it is significant to combine these two disciplines to share the knowledge of creating and using descriptive models.

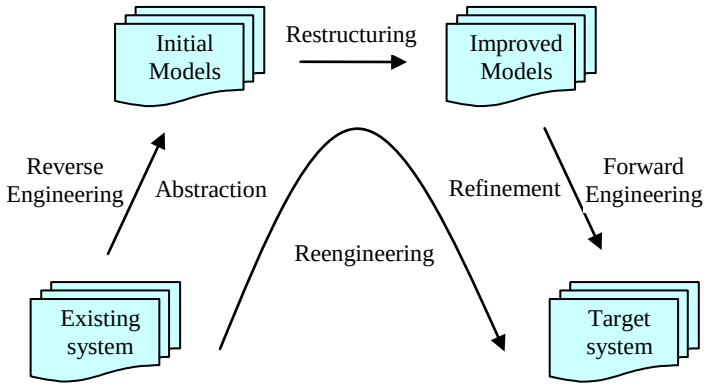



Figure 2-3 Reengineering in Context of MDA [25]

In the context of software modelling, reverse engineering is defined as the process in which software artefacts from legacy systems are restructured through model transformation based on well-defined steps [25].

A number of techniques related to software reengineering are proposed. All of them emphasise on comprehending and reusing the assets of the previous development. Without covering all of them, some terms derived from [183] relevant to software reengineering in the context of MDA (Figure 2-3) are listed as follows which may provide a clear understanding of this domain:

> **Reengineering** *is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. The process of reengineering computing systems involves three main steps: reverse engineering,*

*restructuring, and forward engineering.*

**Forward engineering** *is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.*

**Reverse engineering** *is the process of analysing a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or higher level of abstraction.*

**Program understanding or program comprehension** *is a term related to reverse engineering. Program understanding implies always that understanding begins with the source code while reverse engineering can start at a binary and executable form of the system or at high-level descriptions of the design. Program understanding is comparable with design recovery because both of them start at source code level.*

**Design recovery or reverse design** *is a subset of reverse engineering. Design recovery recreates design abstractions from a combination of code, existing design documentation (if available), personal experience, and general knowledge about problem and application domains.*

**Program Transformation** *is the act of changing one program into another. The term program transformation is also used for a formal description of an algorithm that implements program transformation. The languages in which the program being transformed and the resulting program are written are called the source and target languages, respectively.*

**Model Transformation** *is a mapping of a set of models onto another set of models or onto themselves, which can be broken into two broad categories: model translation and model rephrasing. In the former, a model is transformed into a model of a different language, and in the latter, a model is changed in same modelling language.*

### 2.5.3 Model Driven Security

Manual development of security policies is difficult, even for skilled security and middleware specialists [144]. Using model-centric and generative MDA approach for

development of secure systems can reduce the complexity by using model transformation.

The term “Model Driven Security (MDS)” [12] takes the essence of MDA for applying security aspects to an application and is considered as alternative method to build secure software systems. In MDS, security properties are modelled into the high level system models and thereby security infrastructures can be generated from the built models automatically by using tools.

Though MDE uses security models to develop secure information systems, it is different from those traditional security models in that security models in MDA context are built-in and scattered throughout the high level system models. Under the MDA philosophy, these integrated security models can be transformed into executable models.

There are currently several MDS approaches focusing on the MDE approach for supporting the development of secure systems. These works try to identify any gaps between MDE and security engineering. As described in Section 2.4.2, SecureUML [104] and UMLSec [83, 85, 86] are two famous secure modelling methodologies in MDS. SecureUML mainly focuses on access control constraints based on the RBAC in design models with the proposed security modelling language based on UML extensible mechanism (UML profile). UMLSec is another well-known MDS approach based on UML profiles. Unlike the SecureUML only focusing on authorisation, UMLSec addresses multiple security concerns such as security requirements, threat scenarios, security mechanisms, security concepts etc. However, lack of the automatic transformation from implementation to code is a big miss in UMLSec.

In addition, the SECTET framework for securing web services in MDS is proposed by Alam et al. in [1, 2]. Similar to SecureUML, SECTET mainly addresses RBAC as its security concerns and focuses on generating XACML security infrastructure. MDS application in smart card development is proposed in SecureMDD [124]. In SecureMDD, formal abstract state machine (ASM) and Java Card Code are derived from UML PIMs and are used for generating code. Model driven security application in secure data warehouse (DW) is proposed by Soler et al. [156]. UML profile for modelling security is used as well to create secure PIM. A set of QVT rules are defined

to facilitate the model transformation from secure PIMs to secure PSMs.

## 2.6 Model Slicing

To understand and test a large software product is a very challenging task. One way to ease this is program slicing, which is a technique emphasising on some certain behavioural aspects of a program and removing non-relevant codes to this behaviour of the program [96]. Another is model based slicing [155], which is a technique for decomposing large software architecture model into smaller models to identify relevant model parts and extract related model elements throughout model that corresponds to user defined slicing criterion.

Unified Modelling Language (UML) is widely used to represent and construct the architecture of software system with the help of its various model diagrams. With increase in the size and complexity of software product, UML models representing the high level design of the product tend to become huge and complicated which may comprise thousands of interactions involving hundreds of objects [96]. For better visualisation of software architecture, impact analysis and for test case generation the properties of system architecture with slicing can be taken into account.

Traditional slicing techniques [161, 175, 176] usually focus on analysing the data or control dependency relationships among the program statements. It is not the case when slicing the architectural models of software is performed because there are several other kinds of relationships among models, such as the relationship between class and class, operation and class, operation and operation, object and class, and object and object etc.

While the problem of software program slicing is a well-studied topic, there are relatively few works in slicing for model slicing. There are no studies which have investigated the work practices of model slicing for security. In Chapter 4, a model slicing method for security has been proposed. In this section, an overview of model based slicing will be revisited, including the various general approaches and techniques used to compute slices.

For better understanding, classification method for model slicing in [155] is adopted where research on model slicing is grouped by dependency relationships, control and

data flow, UML/OCL constraints, modelling languages and feature. Taking consideration of the reverse engineering legacy system based on models, using dependency relationship to slice models is chosen in this research.

Researches using dependency relationships can be found in the following works. Zhao describes a new dependence analysis technique in [187] to support the software architecture development. The work is extended by Zhao in [188] in which a static architecture slicing technique is introduced. Wu and Yi [181] develop an approach that comprises different class relationships to define dependency relationships among classes. Van Langehove [166] propose an algorithm to reduce the number of interference dependencies in state charts by using the concept of slicing with concurrent states. Wang et.al [171] present an approach to slicing hierarchical automata for model checking in UML state diagram. Samuel et al. [148] present a methodology to generate dynamic slices and test cases with the help of UML sequence diagram. Lallchandani et al. [95] propose a technique for constructing dynamic slices of UML models using the integrated state-based information. This work is extended by Lallchandani et al. in [96] in which a dynamic model slicing method is proposed based on the proposed intermediate representation named MDG which is constructed from UML class diagram and sequence diagram.

From the discussion above, it can be concluded that several kinds of diagrams are used to model slicing, for example class diagram, control and data flow diagram, sequence diagram, state chart and so on. Different diagram slicing methods may have different features. However, up to now, the models that can be reverse engineered from legacy system are limited. After checking reverse engineering tools which can be used to extract models from source code, class diagram and sequence diagram are chosen as the representation models to be sliced.

## 2.7 Risk Analysis

In risk analysis domain, mathematical techniques are used to calculate several metrics and quantify the security level of a given system. Usually, the metrics includes the likelihood, exposure, and consequences of the occurrences for event relevant to security

[5]. McGraw highlights the need to perform risk analysis at software design level in [111]. According to him, “Design flaws account for 50 percent of security problems, and architectural risk analysis plays an essential role in any solid security program [111]”.

There are methods and tools confirm the state of the art in risk assessment, such as BS7799, MAGERIT and OCTAVE. BS7799 [19] is a British standard related to information security management which is suggested to be used in industry. SP 800-30, Fips 65 [159] is another information security guidance regarding to risk management developed by NIST. MAGERIT is an open methodology for Risk Analysis and Management, developed by the Spanish Ministry of Public Administrations, offered as a framework and guide to the Public Administration [106]. OCTAVE [3] is a heavy-weight risk methodology approach originating from Carnegie Mellon University’s Software Engineering Institute (SEI) in collaboration with CERT. OCTAVE focuses on organisational risk, not technical risk. However, OCTAVE is large and complex, with many worksheets and practices to implement and it does not provide a list of “out of the box” practices for assessing and mitigating web application security risks. CVSS is a complicated scoring system composed of three metric groups developed by the US Department of Homeland Security (DHS) [116]. The disadvantage of CVSS is that it does not find or reduce the attack surface area or help enumerate risks within the target system. Each method and tool has its own benefits. Microsoft proposes a risk management guide to provide the security risk solution in their product [121]. Mehari [114] is another risk analysis assessment and management method developed by CLUSIF (French association of information security professionals) which complies by design to ISO/IEC 27005 guidelines.

Halkidis and Tsantalis [71] propose a risk analysis approach for software systems considering the contained security patterns in the design. In their method, an evaluation is made to measure the effectiveness and extent of specific security pattern shielding from known attacks. Fuzzy set theory and fuzzy fault trees are used as the mathematical model to process the evaluation result and thereby the risks of each type of attacks are computed.

Except for the risk assessment methods and tools discussed above, there are many

methods proposed to figure out the risk assessment for web application [28, 82, 146]. Cock et al. [28] propose a method to address security problems in web applications by modelling security tokens. All the possible threats related to web applications are pointed out in their method. Romero et al. [146] propose a methodological tool for web application focusing on one of the risk assessment steps - asset identification. In [82, 146], STRIDE model is used to perform the risk assessment for web application but lack of threat identification.

## 2.8 Security Pattern

Patterns have been proven successful in many areas of software development, and they appear to be particularly valuable for secure systems development. Security pattern was first proposed by Yoder and Barcalow in [184]. For better understanding of security pattern, its definition derived from [151] is given as follows:

> **A security pattern** *describes a particular recurring security problem that arises in specific contexts, and presents a well-proven generic solution for it. The solution consists of a set of interacting roles that can be arranged into multiple concrete design structures, as well as a process to create one particular such structure.*

The advantages of a pattern approach to security are shown below [151]:

- Patterns codify basic security knowledge in a structured and understandable way
- The pattern representation is familiar to software developers and system engineers, a key portion of their audience
- Because patterns are already used to capture organisation and system engineering knowledge, using patterns to capture security knowledge helps to improve the integration of security into systems and enterprises

Research on security patterns has become an active theme in security domain. A number of security patterns have been proposed for being applied in different contexts and solving different security problems. Other works, such as security pattern classification, organisation, integration, security pattern repository, as well as developing security patterns play key role in security pattern application. In this section, major contributions

to the fields are reviewed.

Analogy to the examination of software design patterns [54], Romanosky [145] addresses security concerns at high level abstraction and proposes security design patterns. The proposed patterns can be used to penetrate multiple layered security concerns and handle the problem of communication with untrusted third-party systems.

Steel et al. publish a book [158] focusing on security patterns for Java web applications. These are design level patterns used for protecting Java platform application with detailed diagrams and sample codes.

Schumacher et al. propose a number of security patterns in their book [151]. The patterns in their book include high level patterns describing the process to secure software development and design level patterns specifying how the detailed security artefacts can be created.

Open Group propose a guide to security patterns in their report [16]. The patterns presented in their report are general purpose patterns range from architectural level patterns to design level patterns and are applicable to software systems implemented using many different languages.

Kienzle et al. present a security patterns repository in the report [90]. The patterns involved in their report can be classified two categories: procedural patterns and design patterns. Procedural patterns emphasise the process to design, implement and configure secure software while design patterns are applicable to how to design and build secure applications.

Besides the above books and reports, many other works on security patterns have been proposed in different contexts. Several papers describe security patterns intended for special purposes, such as security anti-patterns in [92], security patterns for web applications are proposed in [90, 177], security patterns for agent systems [128], security patterns for cryptographic software [17, 101], security patterns for mobile Java Code [107, 158], security patterns for operating systems [48], packet filter and proxy-based patterns for firewall [47, 150], and finally metadata, authentication and authorisation patterns [46, 100].

The increasing number of patterns and similar security patterns appear in the literature with different names make it necessary to develop classifications to security patterns. A classification organises patterns into groups of patterns that share one or many properties such as the application domain or a particular purpose. Many security pattern classification approaches have been proposed since Gamma et al. introduced the first classification of security patterns (GoF patterns) [53].

Heyman et al. [77] classify 220 security patterns into three categories, guidelines, process and core patterns. Design guidelines described by Viega and McGraw in [168] are used to compare 8 security patterns by Cheng et al. in [26]. They extend their classification based on access types of security patterns and thereby classify in the term of application level: network-level, host-level and application-level. Kienzle et al. [89, 90] classify security patterns into two broad categories, structural and procedural. Another broad classification of security patterns is made by Blakley et al. [16] in which two broad category of security patterns is made: available patterns and protected patterns. Halkidis et al. [69] examine the evolution feature of security patterns by comparing the patterns derived from [16]. Laverdiere et al. [99] propose a six sigma method to classify the 12 common security patterns from [26] and [69]. Hafiz et al [65, 66] propose a multi-dimension classification scheme taking consideration of security CIA features, application context, security wheel, McCumber cube, STRIDE threat modelling, and hierarchical classification. The relationships used in their work are similar to the dependencies among security problem patterns suggested by Hatebur et al. [75].

In this section, related work on security patterns has been reviewed which shows lots of security patterns have been proposed with several methods to classify and organise them. Although research on security patterns have become an active topic in the security engineering domain, none of them can directly fulfil the purpose of this research for selecting appropriate security patterns to satisfy the elicited security requirement. Therefore, a security ontology is proposed in this research to smooth the process by properly organising security patterns with a proposed multiple criteria classification method.

## 2.9 Summary

There is very little work concerning the improvement for software security from the evolution perspective. Although several approaches have been proposed to enhance the security for existing software, there is currently no comprehensive method to assist developers in reengineering software for security concerns. Lacking of the support for security improvement for legacy systems is usually seen as the consequence of: (1) security artefacts in the current design being difficult to recognise and isolate from other system aspects; (2) security requirements of the legacy systems being difficult to elicit and analysis; (3) software developer lacking security expertise in improving security level for legacy systems. All these become huge challenge and special concerns when considering complex systems especially web based applications.

Existing approaches are not comprehensive enough in the sense that they focus either on some specific developing stage, such as architecture, design and implementation, or on some specific security aspects, e.g. authentication or authorisation. Moreover, they typically offer no guidance on how they can be integrated into the current design component or system models. The research in this thesis will bridge the gap and provide a comprehensive approach to integrate security expertise into the system design and development.

Most of the existing security engineering methods focus on the methodologies of integrating security features to new development systems. Different from the related studies, this research will integrate the security engineering with software evolution. What this proposed approach does will provide a systematic and effective guidance to implement security evolution process. The current research related to software evolution driven by security has covered a number of domains. In this chapter, the background and related work of SEMDA are introduced:

- A brief introduction of software security engineering is reviewed and the key concepts and features are described.
- A brief discussion on legacy system, software evolution, software reengineering and their definition are described. The relation between software evolution and software reengineering is given, and a general reengineering process of software

systems is represented.

- Software security in software evolution is introduced. Three security engineering processes are reviewed. Detailed discussions are conducted from security requirement and security design point of views.
- Security requirement engineering is introduced and detailed discussion is conducted from the point of view of security requirement elicitation, security requirement analysis and prioritisation.
- Security implementation in design phase is presented with three main methods that are component-based, aspect-oriented and service-oriented security approach.
- A brief description of MDA in the discipline of reengineering is revisited from the point of reverse engineering and model driven security.
- The basic concept and related works of risk analysis are discussed.
- Security patterns, derived from design patterns as the effective solution to the emerging security problems, are discussed along with the relevant researches are reviewed.

# Chapter 3

# Security Driven Software Evolution Approach

**Objectives**

- To summarise the rationale for the proposed approach, SEMDA (**S**ecurity-driven Software **E**volution Using **M**odel **D**riven **A**pproach).
- To introduce SEMDA architecture.
- To describe SEMDA process model.

## 3.1 Overview

For conforming to the upgrading security requirements, a comprehensive security evolution approach called SEMDA (**S**ecurity-driven Software **E**volution Using **M**odel **D**riven **A**pproach) is proposed in this chapter. The SEMDA approach aims to improve the security level for legacy systems from the software reengineering perspective with models as the centric view. Software reengineering is the core technique for successful software evolution which can be seen as a combination of reverse engineering, functional restructuring and forward engineering. Therefore, it is necessary for the proposed SEMDA to include existing well-established techniques and developing relevant approaches to support the successful implementation of security evolution for legacy systems. To achieve the aim, reverse engineering techniques, security requirement engineering techniques and forward engineering techniques have to be used to understand, analyse, evaluate the legacy system and thereby regenerate a security enhanced system.

In following sections, the framework of SEMDA approach is presented in detail. Moreover, various phases, models and techniques, their purpose and their relations are introduced.

## 3.2 Framework of SEMDA Approach

A unified framework is built to support the proposed approach as shown in Figure 3-1, which provides the guidance to the security driven evolution process. The whole process is divided into separate phases, activities and tasks, and structured into different abstraction levels with different system models. More precisely, Figure 3-1 illustrates the architecture (or reference model) for the proposed SEMDA. This architecture is composed of various models and hierarchy analysis methods. Each method computes a model of the system which is then fed to higher level analysis (e.g., model extraction yields the UML diagrams that can be used to compute system partitions).

As shown in Figure 3-1, the whole framework can roughly be divided into three phases, which are a serial of:

- **Legacy System Understanding for Security.** Task of this phase is to reverse engineer legacy systems into UML models, partition the legacy system into subsystems with the help of model slicing technique and detect existing security mechanisms in the legacy system to determine whether or not the provided security in the legacy system satisfies the user's security objectives.
- **Security Requirement Elicitation.** It is the process of analysing key aspects in the legacy systems in terms of security. A new risk assessment method is proposed and used to elicit the security requirements which will generate the detailed security requirements in a specific format to direct the subsequent security enhancement.
- **Security Enhancement.** It is the stage that security patterns as the best practice which are derived from security expertise and fulfilling the elicited security requirements are organised, selected and integrated in the legacy system models with the help of proposed security ontology.

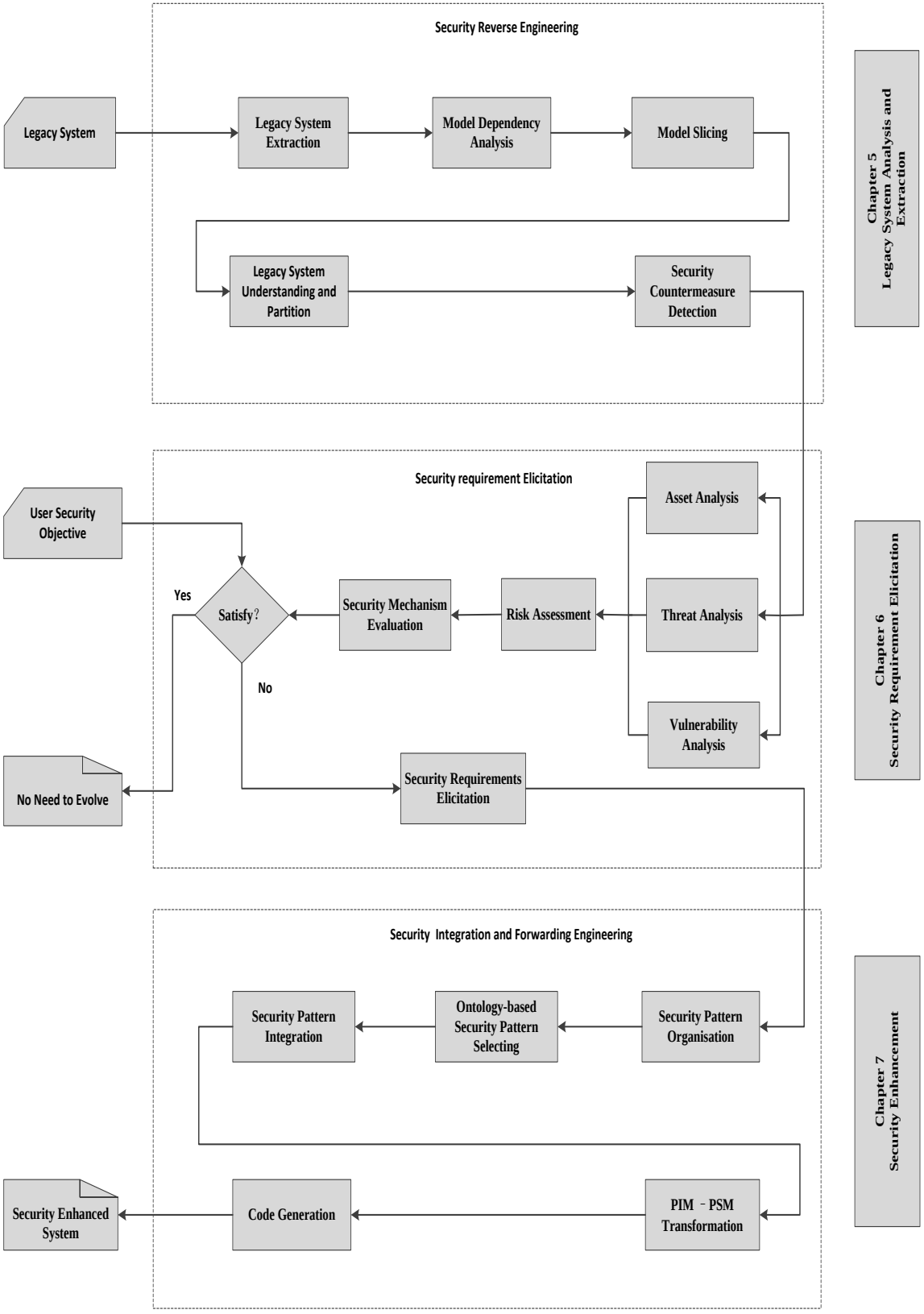



Figure 3-1 Framework of SEMDA Approach

### 3.2.1 Legacy System Understanding for Security

Evolving legacy system is not easy and often encounters many problems due to incomplete or absence of original design information. A successful evolution of legacy systems depends on proper comprehension of their functionalities, contexts and architectures. To achieve this purpose, software reverse engineering techniques are usually used to extract high level diagrams from source codes.

#### 3.2.1.1 Choices of Models

The first step of the proposed framework is to extract higher level models of the legacy system for the later security based transformation purposes. A legacy system has static and dynamic characteristics that display its functionality and represent its structural and behavioural characteristics respectively. The modelling of a legacy system concentrates on the reflection and comprehension of the legacy system at higher level of abstraction. The main purpose is to understand the structure of the target legacy system and its main tasks.

UML has proved to be a good platform for modelling real systems. When modelling a legacy system with UML, the information in the legacy system is refined by using the UML diagrams. Through the extraction of UML diagrams from legacy code, the transformation has realised analysis platform on UML in order to be helpful on the comprehension of legacy systems based on the general analysis language UML.

To evolve an existing legacy system, both static and dynamic information are useful. Static information describes the structure of the software while dynamic information specifying the runtime behaviour. Static analysis along with dynamic analysis for the legacy system contributes to the various software artefacts and their relationships.

UML has static and dynamic modelling advantages. It satisfies the needs of software evolution. At the same time, UML presents a visual description of the system and makes the process of software evolution easily acceptable. Meanwhile, a large number of tools support the transformation from UML diagram to code, and UML facilitates the reusability of software evolution, which is also helpful for forward engineering in the process of reengineering.

#### 3.2.1.2 UML Model Extraction

Software evolution is supported by producing design models from the legacy software. The software evolution approach is useful when building legacy software into high-level information. The extracted models are utilised to get an overall picture of the current state of the legacy software. The dynamic models are used to support tasks such as understanding the current behaviour of the legacy software.

As the first step of whole framework, the task of this phase is to extract UML diagrams from legacy source code. When it comes to reengineering legacy system, source code is thought of as the most reliable part to be modified for satisfying the new requirements. Traditional reengineering methods rely on software representation techniques such as data and control flow diagram, Abstract Syntax Tree (AST), or UML class diagram to represent the various software aspects and the interrelationships between them.

There are two major stages of UML extraction from legacy systems as being structural and behavioural.

- The structural stage contains UML structural or static diagrams extraction. UML 2.0 uses six diagrams to model the static parts of software system, which are class diagram, object diagram, component diagram, deployment diagram, package diagram, composite structure diagram.
- The behavioural stage includes the UML behavioural or dynamic diagrams extraction. UML 2.0 uses seven diagram to model the dynamic parts of the software system, which are communication diagram, timing diagram, use case diagram, sequence diagram, activity diagram, state machine diagram and interaction overview diagram.

In this thesis, class diagram and sequence diagram are chosen to represent static model and dynamic model of the legacy system and extracted by using existing extraction toolset.

#### 3.2.1.3 UML Model Slicing

With the increase in products' sizes and complexities, UML models extracted from the source code are likely to become large and complex. It is possible that hundreds of

objects are involved in thousands of interactions which make the extracted models from such large architecture harder to read and understand. Moreover, it tends to be tedious and poor readability on one side and it is valuable to judge the impact of a certain change of one model elements on other parts on the other side [96].

Different UML diagrams represent different system views. For a better understanding of the extracted UML diagrams, especially for security related analysis, it is necessary to slice the diagrams. Different from program slicing, model slicing aims to reduce the legacy system view at the model level.

In order to slice the UML models, different dependency relationships among classes need to be taken into consideration, which are relation dependency, operation dependency, control dependency, data dependency, call dependency and message dependency. Moreover, an intermediate representation of diagrams is constructed based on the defined class dependency relationship and sliced according to the slicing algorithm and appropriate slice criteria.

#### 3.2.1.4 System Partition

The proposed approach aims to extract the reusable legacy components from the underlying legacy system. In this context, method and process are needed to partition existing systems into notable collections of components, each of which potentially implements an object. However, legacy systems are huge and usually the packaged systems that composed by rich and old structures. To address this problem, a specific decomposition method is developed based on the class dependency analysis and model slicing technique.

There are two main challenges to be taken up, one is how to determine the cohesion degree in a cluster, and the other is what kind of intermediate diagram can be used to facilitate the partition. After examining the dependency relationship among classes and objects in UML diagrams, the proposed CSDG graph is used as the intermediate representation to serve the system partition and each type of edges in CSDG is weighed as the parameter to decide the cohesion degree. Moreover, a decomposition algorithm is proposed to search high independent clusters on the basis of CSDG and independent metric.

#### 3.2.1.5 Security Mechanisms Detection

The security mechanisms detection process is to identify the existing security countermeasures used in the legacy system so as to conduct a fully evaluation towards security level of the target system. A method to conduct security countermeasures detection for legacy system is proposed which is divided into two main parts.

- Extraction phase. Reverse engineering tools are used to extract and gather all relevant information from the legacy system under evaluation.
- Identification phase. This phase inspects each of the gathered information in order to determine whether it is relevant with the security artefacts listed in Security Artefacts Base which stores a list on possible security issues, and is created and maintained by security expert. The results of the identification phase are a mapping list showing whether or not there are any security artefacts in the legacy system and what types of security countermeasures they belong to.

After this stage, system security analyst can make an evaluation if the existing security mechanisms are enough to meet the user's security objectives and based on which a decision can be made whether or not a security evolution is needed.

### 3.2.2 Security Requirement Elicitation

This section focuses on eliciting security requirements by performing risk analysis, which is one of the effective sources to identify security requirements according to the security standard ISO 27002 [80]. The security requirement is derived from assessing risks of the legacy system, taking consideration of the system's overall security objectives. Through the risk assessment, threats threatening the assets are identified. Risk assessment is the process to assess the risk level of the legacy system taking security factors into account in a quantified manner.

For further implementation, security requirement needs to be formatted and represented as 4-tuple <*Asset, Threat, SecurityAttribute, Priority*>. Element *Asset* means every requirement has to be related to one asset, element *Threat* represents possible threats threatening the asset, element *SecurityAttribute* means the features that make an asset valuable, element *Priority* shows the order of development which can be computed

from the proposed risk assessment approach.

#### 3.2.2.1 Asset Analysis

Asset is anything valuable to the organisation. Asset analysis consists of asset identification and asset criticality ranking. Asset is the final target of a security attack. Knowing how many assets in the system, what categories they belong to and the criticality of each asset in the legacy system are important to system security design and security implementation. In the light of security concerns, the assets with sensitive information or process are especially of great importance. The result of assets analysis is an asset list with asset name, category, security feature and criticality.

#### 3.2.2.2 Threat Analysis

Threat is the potential cause of an unwanted event, which may lead to harm to a system. From the asset analysis, what should be protected in the system is determined. The next step is to make sure how to protect the identified assets, that is to make sure what kinds of threats threatening the asset. An environment-driven threat elicitation approach is proposed for web applications taking the consideration of environments where the web application hosts. Quantification towards the identified threats is conducted by using Microsoft's DREAD. The output of threat analysis is a threat list with threat name, violated CIA feature and risk score.

#### 3.2.2.3 Vulnerability Analysis

Vulnerability is a weakness of an asset or control, which may be exploited by a threat. Vulnerability analysis is performed by using existing scanning tools and evaluation scheme CVSS is used to assess the severity of the identified security vulnerabilities.

#### 3.2.2.4 Security Evaluation

From the previous analysis of security mechanism detection and security requirement elicitation, combined with the user's security objectives, a security evaluation can be made by quantifying the effectiveness of detected security mechanisms to the identified security requirements and thereby a conclusion can be drawn after considering the user's security objectives.

### 3.2.3 Security Enhancement

In SEMDA framework, security enhancement phase is one of most important steps during the entire evolution process, whose task is to perform security improvement for the target system on the basis of the abstraction artefacts from the reverse engineering phase in Chapter 4 and in line with the elicited security requirements in Chapter 5.

The approach to improve the security level of legacy system in this thesis is security pattern which is defined as “a security pattern describes a particular recurring security problem that arises in a specific context and presents a well-proven generic scheme for its solution [150]”.

#### 3.2.3.1 Security Pattern

Security patterns are the best practices documented to solve the security problems, which make it possible for security novice to solve the security problems with the proven security expert solutions. In order to support the security evolution for legacy system, security patterns need to be formatted in terms of security related information and organised to facilitate the mapping from the elicited security requirement to the corresponding security patterns.

For the security related format, security pattern is represented as 3-tuple <*Context, Problem, Solution*>, where *Context* means the environment the security problems arise, *Problem* means the threat or attack the security pattern solves and *Solution* provides the approach how security pattern solves the security problem in the given security context.

In this study, thirty-two well-known security patterns are selected to form the pattern repository and organised by the proposed multi-aspectual classification metric including lifecycle, layer, threat type, application context, domain, and security concerns.

#### 3.2.3.2 Security Ontology

After examining the security risks in the legacy system, a set of security requirements have been elicited. There exist a number of security patterns to solve the security problems in the corresponding context. How to associate security requirements with corresponding security patterns is the key of the proposed SEMDA framework.

Ontology is the formal representation of the entities and relationship that exists in some domain. Ontology is useful for representing and inter-relating many types of knowledge. Security ontology is the application of ontology in information security domain.

In order to relate security requirements with security patterns, ontology is used to represent the concept and relationship among security requirement elements and security pattern elements which forms some kinds of security ontology. The proposed security ontology is written in OWL which is the current recommendation of the World Wide Web Consortium (W3C) for exchange of semantic content on the web. By defining security requirement ontology and security pattern ontology properly, this allows to the correct results of the knowledge that can be inferred from the proposed security ontology by applying the corresponding restrictions and axioms.

#### 3.2.3.3 Security Pattern Integration

Generally speaking, security pattern integration is fairly straightforward. The extracted model from legacy system can be annotated according to the elicited security requirement. Security patterns are represented as UML diagrams. The thing to be done is to integrate these UML models according to the security requirement and security pattern, e.g. user account is a kind of asset treated as an element in UML models and needs authenticator pattern to protect against unauthorised access threat. The more detailed integration will not be discussed in this thesis because the integration techniques can be adopted through existing research.

## 3.3 Summary

In this chapter, a unified reengineering approach, SEMDA (**S**ecurity-driven Software **E**volution Using **M**odel **D**riven **A**pproach), is proposed for software evolution in terms of security domain.

The SEMDA Approach mainly contains three stages, which are security reverse engineering stage, security requirement elicitation stage and security enhancement stage. Some domain analysis and assessment methods, software reengineering methods, and intelligent information process methods and techniques are applied to implement

SEMDA projects:

- Legacy system understanding and extraction. This is the process of assessing whether legacy system needs to be evolved according to current user's security objectives. It comprises model extraction, model slicing, system partition, and security mechanism detection stages. The acquired results will be prepared to start next stages.
- Security requirement elicitation. This is the process to elicit security requirement systematically. A risk assessment approach is proposed to satisfy the need which consists of asset analysis, threat analysis and vulnerability analysis. Quantification metrics are proposed to measure the security level for each security requirement as well as whole system. The outcome of this stage is a formatted security requirements list which will be satisfied by next stage research.
- Security enhancement. It is the final stage where security patterns will be selected via proposed security ontology according to the elicited security requirement.
- The proposed approach has been regarded as a semi-automatic process that involves a series of manual work on the representation from domain analysis, and automatic transformation with defined rules can be implemented with toolset support.

# Chapter 4

# Legacy System Understanding for Security

**Objectives**

- To understand the legacy system
- To extract models from legacy system
- To construct intermediate diagram CSDG for slicing.
- To propose a model slicing algorithm based on the CSDG graph
- To present system decomposition method according to independence metric
- To detect existing security mechanisms in the legacy systems

## 4.1 Overview

Security concerns usually scatter many aspects of software system, especially for those based on the web called web applications. As mentioned in Chapter 2, such software systems are faced with lots of security problems such as design defects, vulnerability exploit and lack of security mechanisms etc., which increase the maintenance cost, difficulties and cause huge losses not only to the economy but also to the reputation of the corporations where the systems are applied. Evolving these software systems for security purpose conforms to the needs of system designer and customer.

For a legacy system, if adequate system design documents exist at hand, it will do benefits to security redesign. Unfortunately, with the just opposite, in most cases, these documents are lost or not well documented which is insufficient to be used for redesign.

Under such circumstances, reverse engineering for the legacy system is needed to achieve high level design artefacts from the legacy code.

However, legacy system in real world are usually huge in size, ranging from tens of thousands of Lines of Code (LOC) to millions of LOC. Obviously, the UML models generated from it would be huge. Legacy systems have static and dynamic aspects. The static aspects include software elements and their relationships, while the dynamic aspects mainly concern the sequential events that perform the tasks. In static modelling, high-level elements are found and subsystems or other logically connected software elements may be represented. In dynamic modelling, behaviour descriptions that show interactions among high-level static elements are abstracted.

Dependency analysis is an important way to analyse, understand and maintain program. It reflects the execution sequence and call relationships among program statements and modules. Lots of dependency analysis methods for programs have been proposed. However, the program dependency analysis methods cannot be directly applied to dependency analysis for UML models since UML is a modelling language which is independent of programming language.

When considering security concerns on the system model level, it requires the designers to figure out the relationships between a specific security-related model element and other model elements. When facing with such issues, system designers might skim through software design artefacts or even source code to discover the relationships. However, even for good software documentation, it is not easy to accomplish the task. Model slicing as the slicing technique in model level can meet such requirements by concentrating on certain relevant aspects according to corresponding slicing criteria.

Figure 4-1 depicts the operational framework for this chapter, and each activity in the framework will be elaborated in the following subsections. System design model in UML static and dynamic diagram can be recovered during the model extraction phase by using reverse engineering tools. An intermediate representation graph called Class Scenario Dependency Graph (CSDG) is constructed based on the dependency analysis of class diagram and sequence diagram and then is computed by model slicing according to the specific slicing criteria. The output slices can be used to decompose the

legacy system based on the proposed decomposition algorithm. A system domain model describing the higher level abstraction of system will be generated after the partition. At last, in order to evaluate the security level of legacy system, security countermeasures in the current design are to be detected with the help of defined security artefact base. The output of Chapter 4 is a system domain model as well as security implementation checklist.

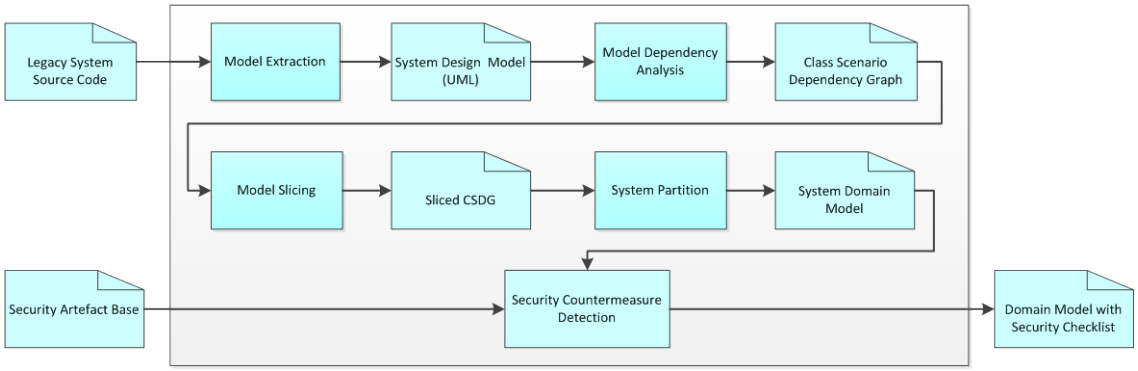



Figure 4-1 Operational Framework for Chapter 4

## 4.2 Legacy System Understanding and Extraction

During the software evolution process, the understanding of existing software (i.e., what the system does) is required to decide what artefacts in the software are going to be modified to comply with the new requirements, software model and computing environments, and how to implement those modifications.

System understanding is a prerequisite for legacy system evolution. It can be regarded as a deductive process of acquiring knowledge about a software artefact through analysis, abstraction and generalisation. It is crucial because legacy systems can be implemented by different building approaches and programming languages, many of them do not have clear specifications.

The adopted analysis can be divided into global analysis and partial analysis in this research. Global analysis is used to identify the business objects, developing solutions and special strategies by the code and architecture design of the legacy system. Partial analysis has been utilised as a process to identify, capture and reorganise the relevant information with the purpose of reusing legacy assets for the migrated systems. A

thorough understanding of legacy system will facilitate the security countermeasure detection and the following security requirements elicitation in Chapter 5.

In this research, the legacy system understanding process is structured as a sequence of activities with the help of reverse engineering tools.

### 4.2.1 Functionality Identification

The objective of functionality identification is to identify and document functionalities in legacy systems. During the process, the analysis primarily targets on the valuable legacy functionalities that may be served as a basis for risk assessment and security countermeasure detection discussed in later sections. Generally, the process of functionality identification is illustrated in Figure 4-2.

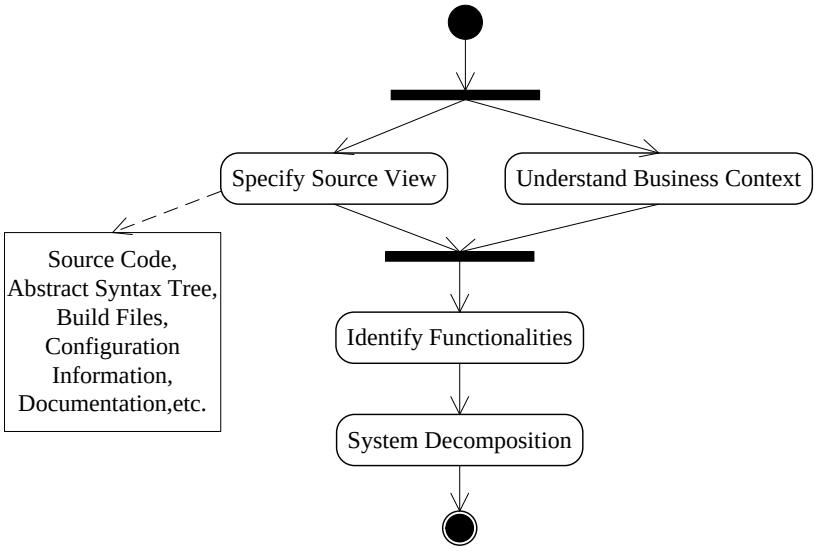



Figure 4-2 Process of Functionality Identification

From this understanding, the process can be divided into two parts: information collection and system decomposition, in which the former one is composed of specifying source view and understanding business context, and the latter one refers to identify functionalities and decompose legacy systems into subsystems according to their correlations.

Developers often encountered a situation that there is very little known about the technical detail of legacy system from the original development team. It turns out that one of crucial tasks at this stage is collecting information and documentation from vary

valuable but incomprehensible codes. The following is a list of relevant activities to accomplish this task:

- Discuss with maintainers
- Skim the documentation
- Observe system operation
- Brief review codes

In certain situations some activities are not applicable due to a lack of resources, such as maintainers have left, documentation becomes inconsistent etc. This is not necessarily a problem because some of these activities may be irrelevant for the goal of reengineering project. However, the absence of resource is a potential risk to the project, and it should be recorded as such in the project report. Developer should plan to keep the report up to date while reverse engineering project progresses and the understanding of the legacy system grows.

Functional identification produces a functional description to legacy systems. Such a description is quite rough and is mainly concerned with the behaviour of the legacy system and not with their structures. However, that may serves as an ideal initial hypothesis to be further refined by applying system decomposition described in Section 4.5.

### 4.2.2 Model Selection

UML has proved to be a good platform for modelling real systems. When modelling a legacy system with UML, the information in the legacy system is refined by using the UML diagrams. Through the extraction of UML diagrams from legacy code, the transformation has realised analysis platform on UML in order to be helpful on the comprehension of legacy systems based on the general analysis language UML.

UML has static and dynamic modelling advantages. It satisfies the needs of software evolution. At the same time, UML presents a visual description of the system and makes the process of software evolution easily acceptable. Meanwhile, a large number of tools support the transformation from UML diagrams to code, and UML facilities the

reusability of software evolution, which is also helpful for forward engineering in the process of reengineering.

UML 2.0 includes thirteen diagrams to improve its modelling quality, shown in Table 4-1. UML 2.0 uses six diagrams to model the static parts of software system, and uses seven diagrams to model the dynamic parts of the software system.

Table 4-1 Static and Dynamic Diagrams of UML [141]

| UML Diagrams | Modelling Analysis |
|---|---|
| class diagram | static modelling |
| object diagram | static modelling |
| component diagram | static modelling |
| deployment diagram | static modelling |
| package diagram | static modelling |
| composite structure diagram | static modelling |
| communication diagram | dynamic modelling |
| timing diagram | dynamic modelling |
| use case diagram | dynamic modelling |
| sequence diagram | dynamic modelling |
| activity diagram | dynamic modelling |
| state machine diagram | dynamic modelling |
| interaction overview diagram | dynamic modelling |

However, in practice, it is not necessary to use all of UML diagrams to model those legacy systems. Some of the UML diagrams are similar. For example, the class diagram

is the most fundamental of the diagrams for modelling the structure of legacy system. An object has the same characteristics as the corresponding class. A class diagram is the abstraction of the common characteristics of the object group. Therefore, if class diagram is used to model the static system structure, the object diagram is superfluous. The behavioural models, such as sequence diagrams and communication diagrams, are used to depict a sequence of actions in an interaction through which a use case is realised [96]. In essence, communication diagram emphasises on which objects interacts with each other while sequence diagrams put more emphasis on the actual order. But they are actually equivalent.

In real systems, the static analysis itself is not adequate for a complete system understanding. For example, class diagram only provides a static view of the class hierarchy. It is impossible to statically discover the real methods and related components that called in an invocation referring to an interface or class. In practice, sometimes this set is still large and reverse engineers hardly want to screen all of them to detect the actual instances. Considering these difficulties, it is necessary to step through dynamic model analysis to learn which components are instantiated at run time and how they interact.

The static model cannot depict the behaviours of objects. While, the dynamic models, on the other hand, cannot adequately represent concerns of structure and dependency relationship. In this thesis, two different UML diagrams are chosen to be reverse engineered from legacy software source code, class diagram and sequence diagram. Class diagram is an important part of UML static modelling in which the classes of the systems and their relationships have been depicted. Class diagram in nature reflects the objects and the static relationship among them in software system. Object diagram instantiates the class diagram. Sequence diagram, on the other hand, is another representation of object diagram which shows the message flow between objects in the software application and also implies the basic associations (relationships) between classes. They have the advantages for visualising how the interactive objects collaborate to perform a job.

Sequence diagrams are the useful tools for system analyser to understand the dynamic behaviour of the software system because they highlight the most important

requirements of a system [94]. Through visualising the execution call traces, sequence diagram can be used to understand the behaviour of the existing system. Moreover, it is a valuable aid to capture scenarios in the software analysis and design phase.

### 4.2.3 UML Model Extraction

Models provide an abstract view of the system, while different diagrams provide concrete representations of the system. Model extraction is the process of generating a model for an existing software system [37]. Extraction of class diagrams of a legacy system can be obtained by identifying the classes and relationships among them, while the reverse engineering of behavioural models lies in analysing the sequence of actions in an interaction among the objects which can help understanding the behavioural aspects of the existing software system. As said in [72] “reverse-engineered sequence diagrams can be created through static or dynamic analysis, the advantages of the latter being increased precision, control over inputs and conditional behaviour, as well as resolution of polymorphism and runtime binding in object-oriented languages”. It is obviously that reverse engineering of sequence diagram is a mentally challenging task. Effective cognitive support can improve the performance of reverse engineering by offloading some or most of the cognitive processing onto an external tool [13].

There are many open sources or commercial tools that can be used to generate UML class diagrams from the source code, supporting most of object oriented language such as C++, C#, Java etc. Some of them can be integrated into the development environment as a plugin. EclipseUML, GreenUML, MaintainJ and Jupe are widely used open source plugin in Eclipse for extracting models from java code. Rational IBM, Microsoft Visual Studio, or Visual Paradigm are some examples of commercial tools with rich functionalities.

Several tools have been developed to aid the reverse engineering for sequence diagram, including Omondo EclipseUML, Rational IBM, Visual Paradigm, ModelGoon, eUML free, MoDisco, ArgoUML, BOUML, Jsonde Call Tracer, covering most object-oriented languages such as Java, C++ and C#.

After comparison with other tools, Visual Paradigm [169] is the most stable one for large scale source code and supports both class diagram and sequence diagram

extraction, thereby it is used to extract the diagrams from source code in this study.

Visual Paradigm for UML (VP-UML) [169] is a powerful UML CASE Tool from the OMG. It can support modelling of UML 2, Business Process Modelling Notation (BPMN) and SysML. Moreover, it provides reverse engineering and forward engineering for java, C# programming language. Class diagram and sequence diagram generation for java is provided and what's more, code generation from class diagram is support as well [180].

Visual Paradigm is a commercial tool for system modelling. It also provides a Community Edition which is free for non-commercial use. Besides that, four commercial editions are provided with different features and prices with 30 days free trial. Even though a community edition for non-commercial use is free, however, it has no reverse engineering and code generation feature. After examination, an Enterprise Edition with 30 days free trial is used to reverse engineering java source code into class diagram and sequence diagram in this thesis. Figure 4-3 shows the extraction of class diagram using VPUML and Figure 4-4 illustrates that of sequence diagram.

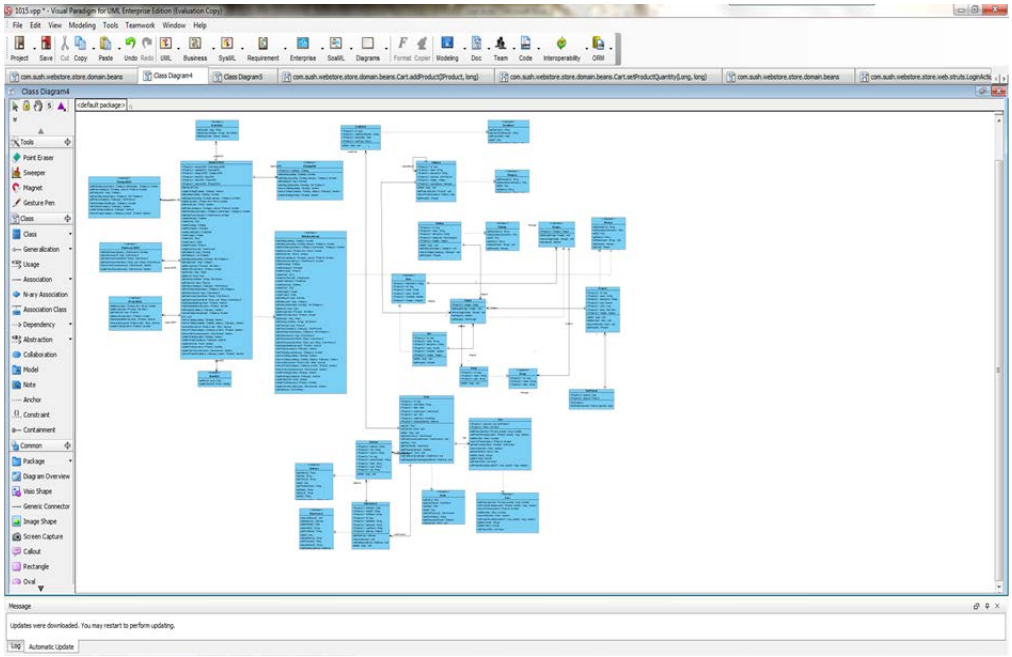


Figure 4-3 Reverse Engineering of Class Diagram using VPUML

Up to now, sequence diagram reverse engineered from source code using automatic tools can not represent the abstract object completely due to the specific implementation

technique of programming language. For example, there may be intermediate result in the extracted diagram, while it should be hidden in the abstract level sequence diagram. Therefore, a manual revision to the extracted sequence diagram is required.

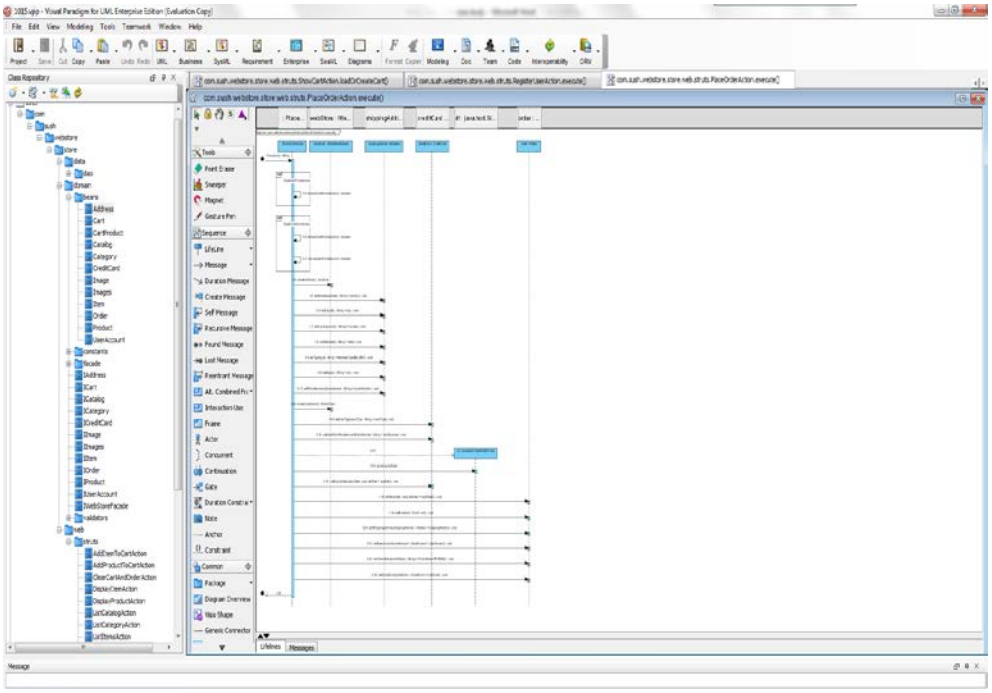


Figure 4-4 Illustration of Sequence Diagram Reverse Engineering using VPUML

## 4.3 UML Model Dependency Analysis

Legacy systems are very likely to outgrow the graphical model. As the complexity of systems keeps increasing, more nodes and edges are added to the diagram which leads to the diagram getting less readability. In the meanwhile, most legacy systems consist of a large number of interdependent elements, such as classes, procedures, data structures, variables and binary modules. These elements are often heavily coupled between each other in the intricate ways, like procedure calls, inheritance relationships and variable references. In such situation, modification at one program segment may affect other modules in unexpected and undesirable ways. Therefore, for the strategy of coping with legacy system evolution, it is necessary to consider the relationship between the functional elements which will be reflected in the relationships in UML models when representing the legacy systems in the higher abstraction view.

### 4.3.1 Representation of Prototype Diagram

As mentioned earlier, class diagram is useful to describe the structure of a system by showing the system's classes, operations, attributes, and the relationships among them. However, on one hand the static model cannot depict the behaviours of objects. Dynamic models, on the other hand, cannot adequately represent considerations of structure and dependency relationships. Therefore, when it comes to representing the system's design artefacts, it is a good choice to take advantages of various kinds of diagrams, such as using class diagram to describe the system's structure while sequence diagram as a complement to illustrate how these objects interact.

Interactions between the objects are performed by messages transferring in object oriented technology. In UML models, the graphical representation of the message is a line segment with an arrow with which the message sender and receiver are connected. The type of message is represented by the arrow type. Both sequence diagram and communication diagram represent the interactions between objects, however, they emphasise the different aspects. Sequence diagram clearly depicts the time sequence among object interaction, but does not indicate the relationship among them. While communication diagram expresses the relationships among objects, but time sequence must be obtained from sequence number.

Thus, in this thesis, the class diagram dependency is analysed with the help of sequence diagram. Since sequence diagram describes execution scenarios of the system, the scenario path will be taken into account in the proposed method.

For the purpose of consistency, the definition of UML class diagram is given below.

**Definition 4.1** UML class diagram $D_{cl}$ is a tuple $D_{cl} = \{CLASS, ATTRIBUTE, OPERATION, RELATION\}$, where

$CLASS = \{c_1, c_2, \ldots, c_n\}$ is the finite set of classes in $D_{cl}$;

$ATTRIBUTE$ is the finite set of attributes in $D_{cl}$;

$ATTRIBUTE = \{\forall c_i.attr\_a \mid a \in ATTRIBUTE, c_i \in CLASS\}$;

$OPERATION$ is the finite set of operations in $D_{cl}$;

$OPERATION = \{\forall c_i.oper\_p \mid p \in OPERATION, c_i \in CLASS\}$;

*RELATION* is the finite set of relationships in $D_{cl}$;

$REALTION = \{\forall\ r \mid r \in REALTION, r = (c_i, c_j), c_i, c_j \in CLASS\}$.

UML class diagram depicts a variety of relationships among classes. In UML class diagram, relations are represented by association line and navigation arrow. Dependency, is a form of association that specifies a dependency relationships between two classes, a dependency is displayed as a dashed line with an open arrow that points from the client model element to the supplier model element. Different from other relations, dependency relation is very useful when describing class relationships with no attributes visible. For example, dependency relation shows whether one packet is aware of the existence of the other packet. The component of one packet does not reference any class, component, interface, method or service of the other packet if there is no dependency relation between them.

As mentioned in the previous section, sequence diagram in nature is a flow chart depicting an execution process. It is obvious that sequence diagram includes control dependency information. Scenarios describe the flow of events when executing a use case and each event flow is called a scenario. Scenario path represents the complete trace of threads execution in sequence diagram and it is the trace of participated objects interaction as well.

As described in [157], "a scenario is a formal description of the flow of events that occur during the execution of a use case instance, and it defines the specific sequence of events between the system and the external actors". For the reasons above, scenario is used from sequence diagram to compensate the limitation of class diagram by reflecting the control dependency in sequence diagram to class diagram.

**Definition 4.2** For a given class diagram $D_{cl} = \{CLASS, ATTRIBUTE, OPERATION, RELATION\}$, a sequence diagram $D_{sq}$ is a trituple = {*OBJECT, LINK, MESSAGE*}, where

*OBJECT* is the set of instance of *CLASS* in $D_{cl}$;

$OBJECT = \{\forall obj \mid obj \text{ is the instance of } c_i, c_i \in CLASS\}$

*LINK is* the set of directed arcs connecting communication objects and it is the

instance of association relationship in *RELATION* in $D_{cl}$;

$LINK = \{\forall\ l \mid l = (obj_i, obj_j), obj_i, obj_j \in OBJECT\}$

*MESSAGE is* a finite set of element representing the interaction between objects which can be either synchronous or asynchronous;

$MESSAGE = \{m_1, m_2, ..., m_n\}$

Message is the only way to communicate between objects in UML interaction diagram which conveys information with the expectation that action will occur. Links are thought as the instances of associations. It is against the rule to create a link in a sequence diagram if there is no relationship (i.e. association, aggregation, or composition) among the objects in the corresponding class diagram. As shows in Figure 4-5, there exists a link between the objects of *POST* and *SALE*. Message flows along with the link. A *total()* operation needs to be defined in *SALE* class if a *total()* message is sent to an instance of *SALE* class. In sequence diagram, messages are represented as arrows around the Links and are labelled with sequence number, names and arguments. The name of the message is the same to the name of object's operation which must exist in the class where the receiving object is instantiated from. The numbers in the sequence diagram show the order of message occurrence. The sequence numbers are nested so that you can differentiate which messages are sent from within other messages.

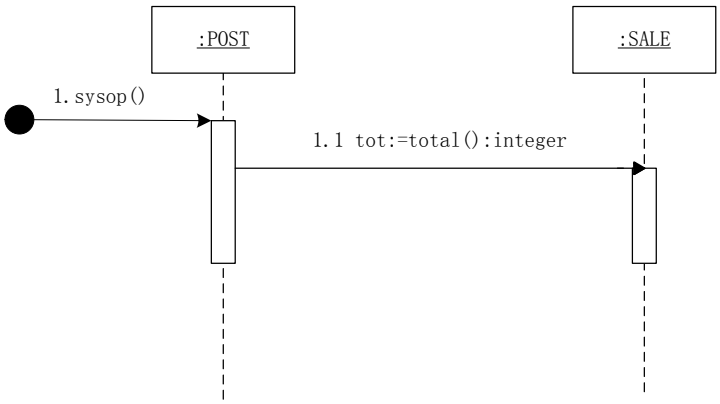



Figure 4-5 Sequence Diagram Example

A message is shown as an arrow line from the sender message end to the receiver message end. A label is used to identify the number, guard condition, iteration or return value of message. Designer can find out the collaborations among objects, trace the

execution path and the variation of messages by identifying the syntax. The objects in the communication can be instantiated from the classes. There exists a corresponding relation in the class diagram for each link between the objects in the sequence diagram. It is exactly based on which can be used to aid the dependency analysis in the proposed method.

### 4.3.2 Dependency Analysis

Program dependence is the dependency relationships between the statements within a program. Data and control flows in the program source code determine the dependency relationship which can be represented by some forms of visualisation using program dependency analysis techniques [181].

In a program with sequence structure, to compute the dependency set for a statement *s* is a process of computing the reachability of the corresponding graph. UML is a kind of modelling language independent of normal programming languages. Therefore, the existing dependency analysis methods for program cannot be applied directly to analyse UML class diagram.

#### 4.3.2.1 Relation Dependency Analysis

There are several kinds of relationships among classes in UML class diagram. The dependences among classes can be defined according to their relationships in class diagram as the following.

- **Association relation**. Association relation is a generic relationship between two classes, which indicates there are some kinds of relation among them but it cannot describe the relation concretely.
- **Aggregation relation**. Aggregation relation is one of special form of association relation. Aggregation represents a “whole-part” relationship, which is known as “has-a” relationship.
- **Composition relation**. Composition is also named as strong aggregation which is known as “owns a” relationship.
- **Generalisation relation**. Generalisation relationship indicates the inheritance

relation and known as "is a" relationship.

- Relation dependency analysis of UML class diagram is the process to analyse how a class is correlated with other classes or how an instance of a class is is correlated with the other class's instances.

**Definition 4.3** Let $s_1$ and $s_2$ be the classes in the UML class diagram $D_{cl}$. If there exists an association navigating from $s_1$ to $s_2$, then $s_1$ is associational dependent on $s_2$, denoted as $\boldsymbol{RD_{ass}(s_1, s_2)}$.

**Definition 4.4** Let $s_1$ and $s_2$ be the classes in $D_{cl}$. If there exists an aggregation relation between $s_1$ and $s_2$, and $s_1$ is the "whole" class while $s_2$ is the "part" class, then $s_1$ is aggregational dependent on $s_2$, denoted as $\boldsymbol{RD_{agg}(s_1, s_2)}$.

**Definition 4.5** Let $s_1$ and $s_2$ be the classes in $D_{cl}$. If there exists a composition relation between $s_1$ and $s_2$, and there is a navigation from $s_1$ to $s_2$, then $s_1$ is compositional dependent on $s_2$, denoted as $\boldsymbol{RD_{com}(s_1, s_2)}$.

**Definition 4.6** Let $s_1$ and $s_2$ be the classes in $D_{cl}$. If $s_1$ inherits structure and behaviour from $s_2$, then $s_1$ is inherited or generalised dependent on $s_2$, denoted as $\boldsymbol{RD_{gen}(s_1, s_2)}$.

#### 4.3.2.2 Control Dependency Analysis

Because of the limitation of class diagram, there are only static structure information of classes can be shown in class diagram. Control dependency cannot be analysed solely on class diagrams, however, it can be performed with the help of sequence diagram.

The UML specification describes a precondition as: "an optional set of constraints specifying what must be fulfilled when the behaviour is invoked [134]." For sequence diagram, guard condition is a kind of precondition.

For better understanding of control dependency analysis, a set of dependency relations is defined. For uniformity, $c_i.oper\_p_i$ represents an operation of $c_i$ while an *oper_p* without class identifier represents any operation in the given class diagram and *attr_a* without class identifier represents any attribute in the given class diagram.

**Definition 4.7** Let $\Gamma = < c_1.oper\_p_1, \cdots, c_i.oper\_p_i, \cdots, c_j.oper\_p_j, \cdots, c_n.oper\_p_n >$ be the operation sequence under a specific use case scenario, then $\xi < c_i.oper\_p_i , c_j.oper\_p_j >=$

$< c_i.oper\_p_i,\cdots, c_j.oper\_p_j >$ is the fragment of ordered operation sequence from $c_i.oper\_p_i$ to $c_j.oper\_p_j$.

**Definition 4.8** For a given UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*} and an operation sequence $\Gamma =< c_1.oper\_p_1,\cdots, c_i.oper\_p_i,\cdots, c_j.oper\_p_j,\cdots, c_n.oper\_p_n>$ from a sequence diagram $D_{sq}$ ={*OBJECT, LINK, MESSAGE*}, control dependency exists if and only if either one of the following conditions holds.

- For $c_i.oper\_p_i$, $c_j.oper\_p_j \in$ *OPERATION*, if $c_i.oper\_p_i$ exists in the $<c_{i+1}.oper\_p_{i+1},\cdots, c_j.oper\_p_j,\cdots, c_n.oper\_p_n>$ and the precondition of $c_j.oper\_p_j$ includes $c_i.oper\_p_i$, then $c_j.oper\_p_j$ is control dependent on $c_i.oper\_p_i$, denoted as $\boldsymbol{CD(c_j.oper\_p_j, c_i.oper\_p_i)}$.
- For an operation *oper_p* ∈*OPERATION*, an attribute *attr_a* ∈ *ATTRIBUTE*, if whether or not *oper_p* can be executed depends on the value of *attr_a*, an attribute of a class, and then *oper_p* is control dependent on *attr_a*, denoted as ***CD(oper_p, attr_a)***.
- Let $object_i$, $object_j \in$ *OBJECT* be the instances of classes ∈ *CLASS*, if whether or not $object_j$ can be executed depends on the execution of $object_i$, then $object_j$ is control dependent on $object_i$, denoted as $\boldsymbol{CD(object_j, object_i)}$.

In non-concurrent event flow, it is obvious to conclude that control dependencies among operations are transitive.

**Rule 4.1** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, combined with an operation sequence from a sequence diagram $D_{sq}$ = {*OBJECT, LINK, MESSAGE*}, if $oper\_p_i$ is control dependent on $oper\_p_j$ and $oper\_p_j$ is control dependent on $oper\_p_k$, then $oper\_p_i$ is control dependent on $oper\_p_k$.

$$CD(oper\_p_i, oper\_p_j) \wedge \mathrm{CD}(oper\_p_j, oper\_p_k) \Rightarrow CD(oper\_p_i, oper\_p_k)$$

### 4.3.2.3 Data Dependency Analysis

Data dependency is such a situation in which the execution of a statement is dependent on the value of some relevant operations. In UML class diagram, data dependency means the value of a variable defined by an attribute or operation is referred by another

attribute or operation.

In security domain, data as an important information carrier should be treated as assets and kept confidential, integrated and available so as to immune from security threat. As described in Chapter 2, CIA is a widely used metric for measuring of information systems security, emphasising on the three key security properties of confidentiality, integrity and availability of information. Confidentiality is related to the border concept of data privacy and refers to limiting information access and disclosure to authorised users. Data integrity, namely, that data should remain integrity and have not been modified inappropriately. To analyse read dependency between attribute and operation among objects of classes will lay a foundation of data confidentiality analysis, similarly, to analyse write dependency between attribute and operation will contribute to data integrity analysis.

**Definition 4.9** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, for an operation *oper_p* ∈ *OPERATION*, let

- *CALL(oper_p)* be the *set* of operations directly or indirectly called by *oper_p*
- *RA(oper_p)* be the set of attributes that are directly or indirectly read by *oper_p*
- *WR(oper_p)* be the set of attributes that are directly or indirectly write by *oper_p*

**Definition 4.10** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, let *c* be a class, *c*∈*CLASS*, then

- If *oper_p* ∈ *OPERATION* and *attr_a* ∈ *RA(oper_p),* then *oper_p* is read dependent on attr_a, denoted as $DD_{rd}$*(oper_p, attr_a),* short as $\boldsymbol{DD_{rd}}$.
- If *oper_p* ∈ *OPERATION* and *attr_a* ∈ *WA(oper_p),* then attr_a is write dependent on *oper_p*, denoted as $DD_{wr}$*(attr_a , oper_p),* short as $\boldsymbol{DD_{wr}}$.
- If $oper\_p_i$, $oper\_p_j$ ∈ *OPERATION* and $oper\_p_j$ ∈ *CALL*($oper\_p_i$), then $oper\_p_i$ is call dependent on $oper\_p_j$, denoted as $DEP_{cal}(oper\_p_i, oper\_p_j)$, short as $\boldsymbol{DEP_{cal}}$.

**Definition 4.11** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, for any *oper_p* ∈ *OPERATION*, the following sets are

defined:

- *Def*(*oper_p*)={*v*| *v* is a variable and defined in *oper_p* as a parameter}
- *Ref*(*oper_p*)={ *v*| *v* is a variable and referenced in *oper_p*}

For a given UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION}*, let $oper\_p_i$ and $oper\_p_j$ be the operations, for $w \in Def(oper\_p_j) \cup Ref(oper\_p_j)$, $v \in Def(oper\_p_i)$, $oper\_p_j$ is data dependent on $oper\_p_i$ if and only if *w* is affected by *v* and there exists $\xi < oper\_p_i , oper\_p_j >$, denoted by $\boldsymbol{DD_{oo}\ (oper\_p_j , oper\_p_i, w,v)}$, short as $DD_{oo}$.

**Rule 4.2** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, for $oper\_p_i , oper\_p_j \in OPERATION$, then

$$\exists v\ (v \in Def(oper\_p_i) \cap Ref(oper\_p_j)) \Rightarrow\ DD_{oo}\ (oper\_p_j , oper\_p_i, v, v)$$

**Rule 4.3** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, the data dependencies between operations are transitive. That is for $oper\_p_i , oper\_p_j, oper\_p_k \in OPERATION$, then

$$DD_{oo}\ (oper\_p_i , oper\_p_j) \wedge DD_{oo}\ (oper\_p_j , oper\_p_k)\ \Rightarrow\ DD_{oo}\ (oper\_p_i , oper\_p_k)$$

From the Definition 4.9 and 4.10, it can be concluded that $DD_{oo}$ is one type of $DEP_{cal}$.

### 4.3.2.4 Member Dependency Analysis

In a class diagram, the attribute and operation are the inherent elements of a class. However, when it comes to analyse the different dependency relationships among various classes in the class diagram, it is necessary to represent this kind of relationships in a formal and clearly way which is the basis of the subsequent model slicing.

**Definition 4.12** Given a UML class diagram $D_{cl}$ = {*CLASS, ATTRIBUTE, OPERATION, RELATION*}, let *c* be a class, $c \in CLASS$, $oper\_p \in OPERATION$, $attr\_a \in ATTRIBUTE$, there exists a member dependency if any one of the following conditions holds:

- If *oper_p* is one of the operation of *c*, denoted by $\boldsymbol{DEP_{mem}\ (oper\_p, c)}$
- If *attr_a* is one of the attribute of *c*, denoted by $\boldsymbol{DEP_{mem}\ (attr\_a, c)}$

# 4.4 UML Model Slicing

Various kinds of dependencies existing among different model elements are usually considered when determining the impact of a change made in the system architecture [96]. This analysis may become even more complex when security concerns are considered among objects due to their cross-cutting feature in nature.

Slicing for software in the model level needs to consider various classes and their relationships, objects and their interactions which have been considered in the previous section. An intermediate representation for UML models is the prerequisite for UML model slicing which is proposed and named as Class Scenario Dependency Graph (CSDG) in this section.

## 4.4.1 Class Scenario Dependency Graph (CSDG)

### 4.4.1.1 Framework of CSDG Method

This section presents an intermediate representation of the extracted UML models, named CSDG which lays a foundation for the subsequent model slicing.

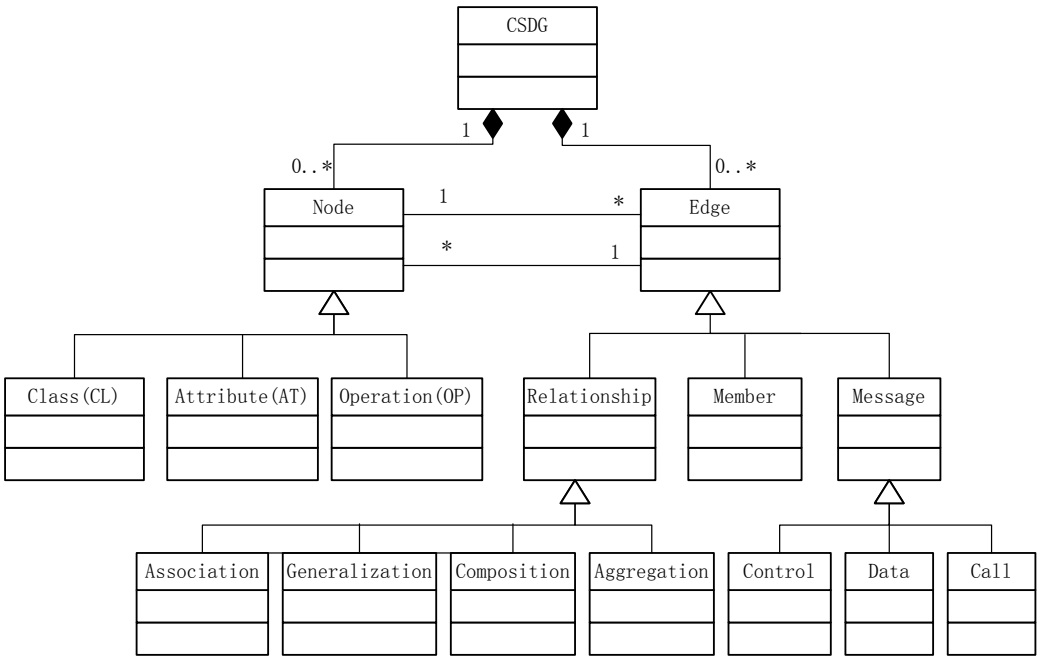



Figure 4-6 Class Diagram of CSDG Meta-model

A class diagram of CSDG meta-model is shown in Figure 4-6. The structural design of

CSDG is illustrated with the elements involved which can be roughly classified into two categories, node and edge. Each instance of CSDG consists of a set of nodes along with a set of edges describing the different kinds of dependencies shown in Figure 4-6.

The various CSDG nodes are listed as：

- Class nodes. Class are represented by $CL_1 \ldots CL_i$.
- Attribute nodes. The attributes of a class *CL* are represented by $AT_1 \ldots AT_m$.
- Operation nodes. The operations called by a class *CL* are represented by $OP_1 \ldots OP_n$.

The edges representing the various kinds of dependencies relationships among the above nodes can be listed as:

- Member dependency edge. Member dependency edges represent the composition of operations, attributes and class which means that *AT* node and *OP* node are the member of *CL* node.
- Relationship dependency edge. Relationship dependency edges represent the way of how classes and instances of classes are interrelating with each other. CSDG represents the class relationships in the same way to its corresponding class diagram which can be classified into association, generalisation, composition and aggregation.
- Message dependency edge. Message dependency edges represent messages flows among the objects which can be classified into the following types according to different message transferring.
    - Data dependency edge. Data dependency edges represent the flows of data where arise the class operation. There exists the data dependency if the parameter and return values of the operation directly or indirectly make use of the class attribute.
    - Control dependency edge. Control dependency edges exist when whether or not the operation of an object is executed is determined by another operation of the object or another object.

- Call dependency edge. Call dependency edges represent the flow of operation calls invoked by objects.

#### 4.4.1.2 Definition of CSDG

For better understanding of CSDG, a graphical method to represent the dependencies in UML diagram is given in this section.

**Definition 4.13** For any class diagram $D_{cl}$ = {*CLASS*, *ATTRIBUTE*, *OPERATION*, *RELATION*} and its sequence diagram $D_{sq}$ = {*OBJECT*, *LINK*, *MESSAGE*}, the corresponding CSDG is a directed graph, can be represented as *CSDG*(*G*) = (*N*, *E*), where *N* is the set of nodes and *E* is the set of edges. $N = CL \cup AT \cup OP$, $E = E_{mem} \cup E_{ass} \cup E_{agg} \cup E_{com} \cup E_{gen} \cup E_{cd} \cup E_{rd} \cup E_{wr} \cup E_{oo} \cup E_{cal}$, for $attr\_a \subseteq ATTRIBUTE$, $oper\_p \subseteq OPERATION$, where:

$CL = CLASS = \{\mu \mid \mu \in CLASS\}$;

$AT = ATTRIBUE = \{\mu \mid \mu \in c.attr\_a, c.attr\_a \subseteq ATTRIBUTE, c \in CLASS\}$;

$OP = OPERATE = \{\mu \mid \mu \in c.oper\_p, c.oper\_p \subseteq OPERATION, c \in CLASS\}$;

$E_{mem} = \{(\mu_1,\mu_2) \mid \mu_1 \in AT \vee \mu_1 \in OP \wedge \mu_2 \in CL \wedge DEP_{mem}(\mu_1,\mu_2)\}$;

$E_{ass} = \{(\mu_1,\mu_2) \mid \mu_1 \in CL \wedge \mu_2 \in CL \wedge RD_{ass}(\mu_1,\mu_2)\}$;

$E_{agg} = \{(\mu_1,\mu_2) \mid \mu_1 \in CL \wedge \mu_2 \in CL \wedge \mu_1 \neq \mu_2 \wedge RD_{agg}(\mu_1,\mu_2)\}$;

$E_{com} = \{(\mu_1,\mu_2) \mid \mu_1 \in CL \wedge \mu_2 \in CL \wedge \mu_1 \neq \mu_2 \wedge RD_{com}(\mu_1,\mu_2)\}$;

$E_{gen} = \{(\mu_1,\mu_2) \mid \mu_1 \in CL \wedge \mu_2 \in CL \wedge \mu_1 \neq \mu_2 \wedge RD_{gen}(\mu_1,\mu_2)\}$;

$E_{cd} = \{(\mu_1,\mu_2) \mid \mu_1 \in OP \wedge \mu_2 \in OP \wedge CD(\mu_1,\mu_2)\}$;

$E_{rd} = \{(\mu_1,\mu_2) \mid \mu_1 \in AT \wedge \mu_2 \in OP \wedge DD_{rd}(\mu_1,\mu_2)\}$;

$E_{wr} = \{(\mu_1,\mu_2) \mid \mu_1 \in OP \wedge \mu_2 \in AT \wedge DD_{wr}(\mu_1,\mu_2)\}$;

$E_{oo} = \{(\mu_1,\mu_2) \mid \forall v, w \in AT \bullet \mu_1 \in OP \wedge \mu_2 \in OP \wedge DD_{oo}(\mu_1,\mu_2,v,w)\}$;

$E_{cal} = \{(\mu_1,\mu_2) \mid \mu_1 \in OP \wedge \mu_2 \in OP \wedge DEP_{cal}(\mu_1,\mu_2)\}$;

From the description of message dependency edge, it can be concluded that:

$$E_{mes}=\{(\mu_1,\mu_2)|\ \mu_1 \in OP \wedge \mu_2 \in OP \wedge DEP_{cal}(\mu_1,\mu_2) \vee CD(\mu_1,\mu_2) \vee DD_{oo}(\mu_1,\mu_2)\}$$

For any pair of nodes $\mu_1, \mu_2 \in N$, edge $(\mu_1, \mu_2) \in E$, $\mu_1$ is a predecessor of $\mu_2$ and $\mu_2$ is a successor of $\mu_1$, thereby there exists an associated edge from $\mu_1$ to $\mu_2$, if any one of the following conditions holds in the CSDG of class diagram $D_{cl}$:

- $\mu_1$ is either *OP* node or *AT* node and $\mu_2$ is *CL* node, then there exists a member dependency $E_{mem}$ between them.
- Both $\mu_1, \mu_2 \in CL$, then there exist a relationship dependency $E_{ass}$, $E_{agg}$, $E_{com}$, *or* $E_{gen}$ between them.
- One of the nodes is *OP* node, the other is either an *AT* node, or an *OP* node, and there exists a data dependency $E_{rd}$, $E_{wr}$ *or* $E_{oo}$ between them.
- A message dependency edge can exist due to one of the following:
  - Both nodes of $\mu_1, \mu_2$ are *OP* nodes, and there exists a call dependency $E_{cal}$ between them.
  - Both nodes of $\mu_1, \mu_2$ are *OP* nodes, and there exists a control dependency $E_{ctl}$ between them.
  - Both nodes of $\mu_1, \mu_2$ are *OP* nodes, and there exists a data dependency $E_{oo}$ between them.

**Definition 4.14** For any pair of nodes $\mu_1, \mu_2 \in N$ in the given UML class scenario diagram CSDG = ($N$, $E$), and there exists $\xi <\mu_1, \mu_2>$, then

$$RD(\mu_1,\mu_2)= RD_{ass}(\mu_1,\mu_2) \vee RD_{agg}(\mu_1,\mu_2) \vee RD_{com}(\mu_1,\mu_2) \vee RD_{gen}(\mu_1,\mu_2)$$

$$DD(\mu_1,\mu_2)=DD_{rd}(\mu_1,\mu_2) \vee DD_{wr}(\mu_1,\mu_2) \vee DD_{oo}(\mu_1,\mu_2)$$

$$MD(\mu_1,\mu_2) =CD(\mu_1,\mu_2) \vee DEP_{cal}(\mu_1,\mu_2) \vee DD_{oo}(\mu_1,\mu_2)$$

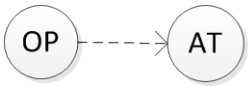
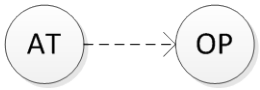
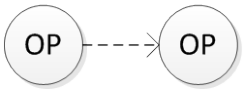
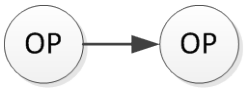
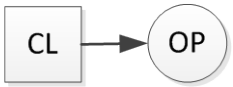
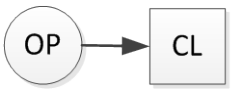
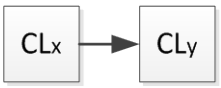
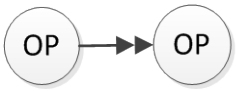
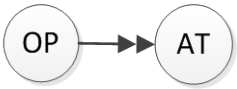
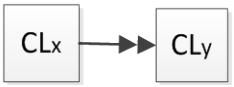
**Definition 4.15** For any pair of nodes $\mu_1,\mu_2 \in N$ in the given UML class scenario diagram CSDG=($N$,$E$), and there exists $\xi <\mu_1, \mu_2>$, then $\mu_1$ depends on $\mu_2$ if and only if there exists $RD(\mu_1, \mu_2) \vee DD(\mu_1, \mu_2) \vee MD(\mu_1, \mu_2) \vee DEP_{mem}(\mu_1, \mu_2)$, denoted as $DEP(\mu_1, \mu_2)$.

### 4.4.1.3 Semantics Analysis of Dependency in CSDG

In the description of CSDG, the semantics of edges depend on the pair of nodes and the specific scenario. The same notations to [96] are used to represent the dependency semantic in CSDG. The edge is represented using a pair of nodes (from-node, to-node). The meaning of the from-node is that the dependency starts with from-node and ends with to-node. The direction from the from-node to the to-node depicts the dependency in the CSDG. The various dependencies semantics in CSDG and their corresponding notations are depicted in Table 4-2.

Table 4-2 Dependency Semantic in CSDG

| Dependency Semantics in CSDG | Description |
|---|---|
| $DEP_{mem}(AT_x, CL_x)$ $DEP_{mem}(OP_x, CL_x)$   Member dependency | Member dependency:<br>• Attributes and operations are members of a class |
| $RD_{ass}$ $(CL_x, CL_y)$ $RD_{agg}$ $(CL_x, CL_y)$   $RD_{com}$ $(CL_x, CL_y)$ $RD_{gen}$ $(CL_x, CL_y)$   Relationship dependency | Relationships between a pair of classes:<br>• Association<br>• Aggregation<br>• Composition<br>• Generalisation |
| $DD_{rd}(OP, AT)$ $DD_{wr}(AT, OP)$ | Data dependency:<br>• An operation reads an attribute |

| | |
|---|---|
|   <br>$DD_{oo}$(OP, OP)<br>  <br>Data dependency | value<br>• An operation writes an attribute value<br>• An variable in a operation is affected by the variable in another operation |
| $DEP_{cal}$(OP, OP) $DEP_{cal}$(CL, OP)<br>   <br>$DEP_{cal}$(OP, CL) $DEP_{cal}$($CL_x$, $CL_y$)<br>   <br>Call dependency | Call dependency:<br>• An operation invocation involves invoking another operation<br>• An object calls an operation<br>• An operation calls an object<br>• An object instances another object |
| CD(OP, OP) CD(OP, AT)<br>   <br>CD(OP, CL) CD($CL_x$, $CL_y$ )<br>  <br>Control dependency | Control dependency:<br>• An operation invocation is in the precondition of another operation invocation<br>• An operation invocation is determined by an attribute value of a class<br>• The execution of an object is determined by another object |

#### 4.4.1.4 Algorithm for CSDG Construction

**CSDG(N,E) Construction Algorithm**

| | |
|---|---|
| Input: | UML class diagram $D_{cl}$ = *{CLASS, ATRRIBUTE, OPERATION, RELATION}*<br>UML sequence diagram $D_{sq}$ = *{OBJECT, LINK, MESSAGE}* |
| Output: | CSDG |
| Initialisation | CSDG(N, E)=NULL |

Procedure ConstructionCSDG($D_{cl}$, $D_{sq}$) return CSDG

1. for every class (i←1 to n ) in $D_{cl}$ do

{*identify class node *CL* from class diagram*}

2. add $class_i$ as $CL_i$ node to CSDG;

3. for every attribute (j←1 to m) and operation (k←1 to x) in $class_i$ $CL_i$ do

{*identify nodes *AT* and *OP* from class diagram*}

4. add $attribute_j$ as $AT_j$ node to CSDG;

5. add $operation_k$ as $OP_k$ node to CSDG;

{* add member dependency edge $E_{mem}$ to CSDG*}

6. add member dependency edge $E_{mem}$ $(AT_j,CL_i)$ to CSDG;

7. add member dependency edge $E_{mem}$ $(OP_k,CL_i)$ to CSDG;

8. end for

9. end for

{* add relationship dependency edge $E_{ass}$, $E_{agg}$, $E_{com}$ and $E_{gen}$ to CSDG*}

10. for every $CL_i$ (i←1 to n) node in CSDG do

11. for every $CL_j$(j←1 to n) node in CSDG do

12. identify the relationship dependency between $CL_i$ and $CL_j$ from $D_{cl}$
13. switch (type of edge($CL_{i,}$ $CL_j$)) do
14. case($RD_{ass}$($CL_{i,}$ $CL_j$)): do
15. add association relationship dependency edge $E_{ass}$($CL_{i,}$ $CL_j$) to CSDG;
16. end case;
17. case($RD_{agg}$($CL_{i,}$ $CL_j$)): do
18. add aggregation relationship dependency edge $E_{agg}$($CL_{i,}$ $CL_j$) to CSDG;
19. end case;
20. case($RD_{com}$($CL_{i,}$ $CL_j$)): do
21. add composition relationship dependency edge $E_{com}$($CL_{i,}$ $CL_j$) to CSDG;
22. end case;
23. case($RD_{gen}$($CL_{i,}$ $CL_j$)): do
24. add generalisation relationship dependency edge $E_{gen}$($CL_{i,}$ $CL_j$) to CSDG;
25. end case;
26. end switch
27. end for
28. end for

{* identify message dependency by examining sequence diagram $D_{sq}$ *}

29. for every $CL_i$ (i←1 to n) node in CSDG do
30. identify the instance of $CL_i$ node in $D_{sq}$ and mark $CL_i$;
31. for (every marked $CL_i$) do

32. identify each interacting object (class) to which an instance of class $CL_i$ sends a message ;

{*these may require reference attributes within the class, depending upon the visibility*}

33. identify every message received by an instance of $CL_i$;

{*these will be the methods for that class*}

34. if (there exists any message) do

35. for each message $m_k$ (k←1 to x) do

36. analyse the type of the message $m_k$ ;

37. switch(type of message $m_k$) do

38. case(CD(*OP*, *OP*)||CD(*OP*,*AT*)||CD(*CL*,*CL*)): do

39. add control dependency edge $E_{cd}$ to CSDG;

40. end case;

41. case($DD_{rd}$(*OP*, *AT*)||$DD_{wr}$(*AT*, *OP*)|| $DD_{wr}$(*OP*, *OP*)): do

42. add data dependency edge $E_{dd}$ to CSDG;

43. end case;

44. case($DEP_{cal}$(*OP*, *OP*)||$DEP_{cal}$(*CL*, *OP*)||$DEP_{cal}$(*OP*, *CL*) || $DEP_{cal}$(*CL*, *CL*)):do

45. add call dependency edge $E_{cal}$ to CSDG;

46. end case;

47. end switch

48. end for

49. end for

50. end for

51. return CSDG;

Note: the shapes of line and arrow of different dependency edge can be found from Table 4-2

List 4-1 CSDG Construction Algorithm

In this section, the proposed method is illustrated with an example referred from [96]. Figure 4-7 illustrates the class diagram of the example case. Figure 4-8 and Figure 4-9 are two sequence diagrams depicting the interactions among the objects of the given example.

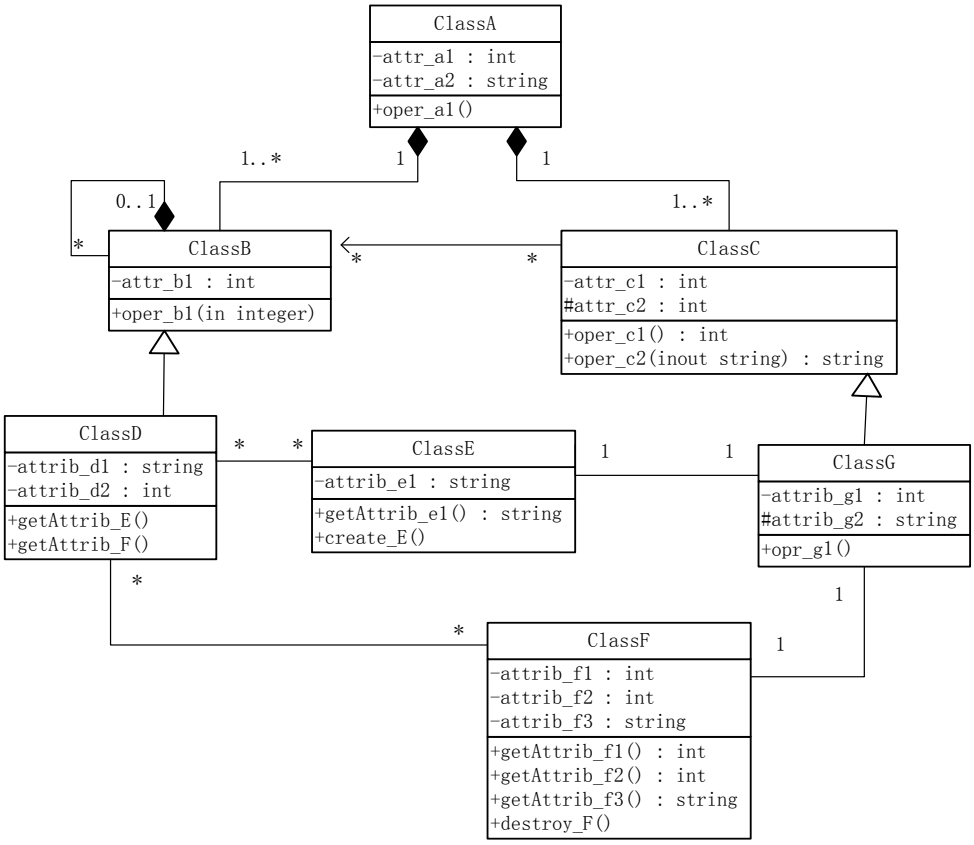



Figure 4-7 Class Diagram of an Example UML Model [96]

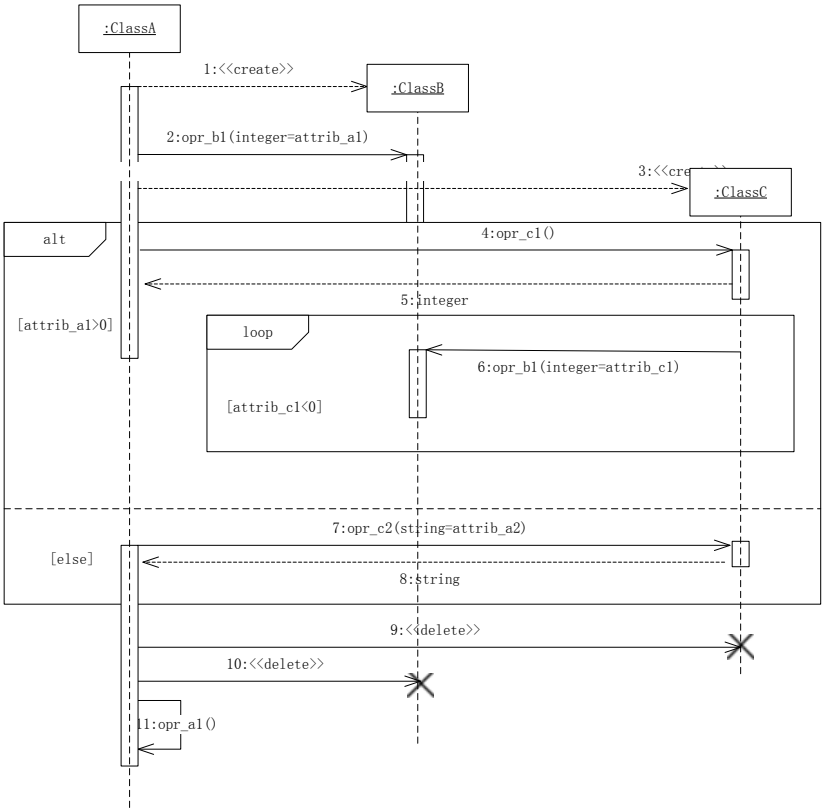



Figure 4-8 Sequence Diagram I of the Example UML Model [96]

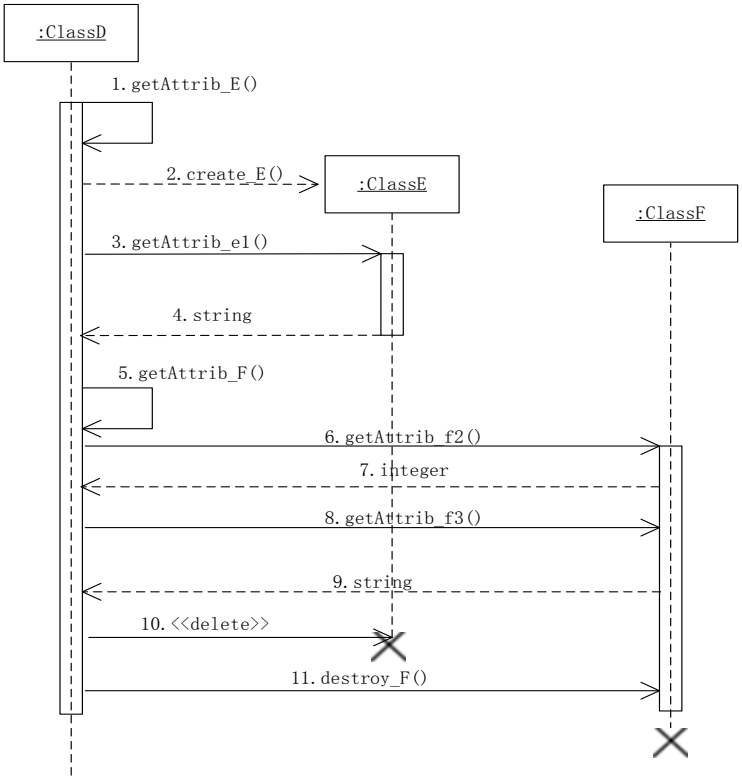



Figure 4-9 Sequence Diagrams II of the Example UML Model [96]

CSDG of the given example can be constructed by using the proposed construction algorithm, and the result is shown in Figure 4-10.

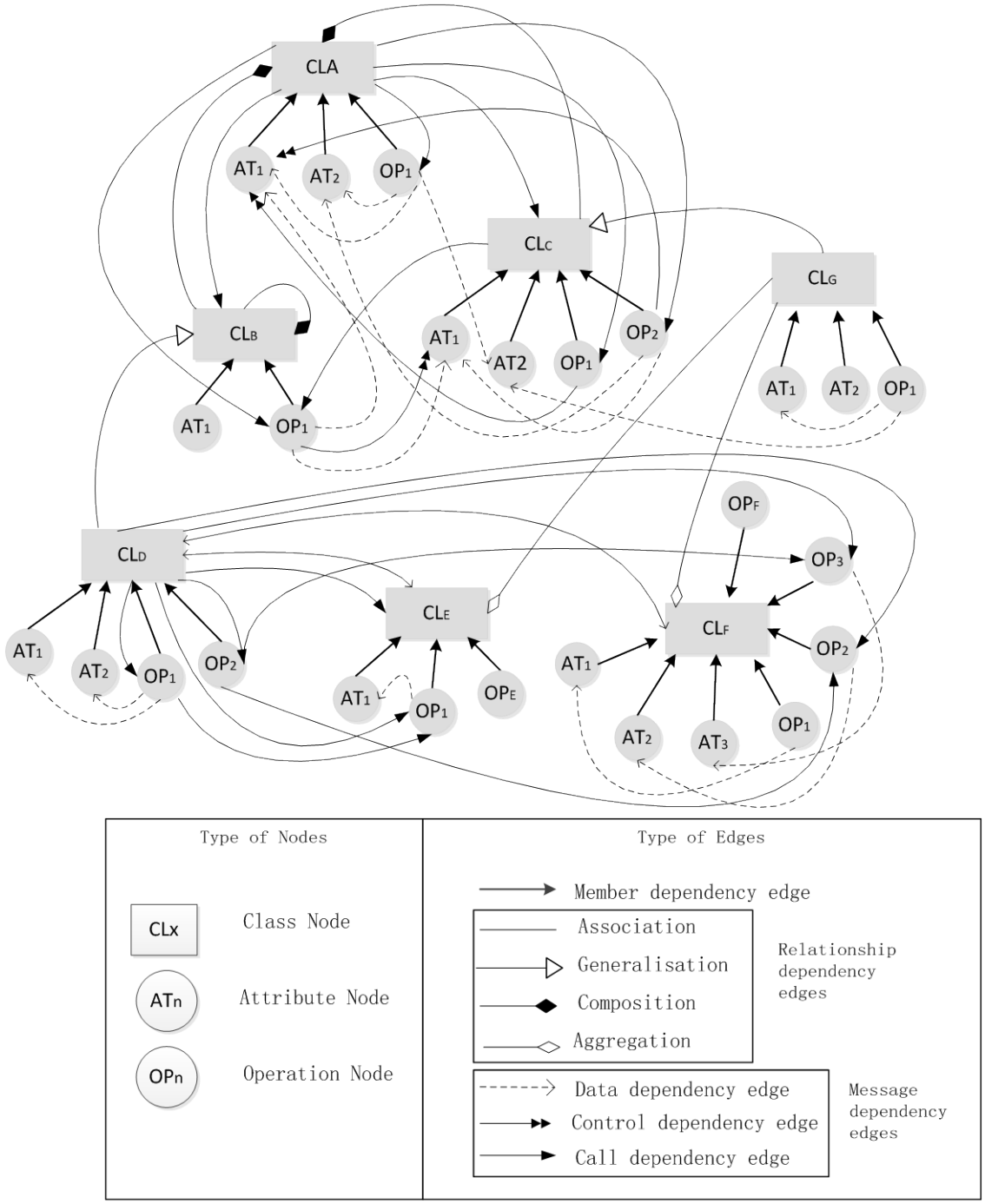



Figure 4-10 CSDG of the Example UML Model

### 4.4.2 Slicing Algorithm

Slicing algorithm is presented in the form of pseudo code as follows.

| **Compute Slice Algorithm** | |
|---|---|
| Requirement | UML class diagram $D_{cl}$={*CLASS, ATRRIBUTE, OPERATION, RELATION*},<br>UML communication diagram $D_{sq}$={*OBJECT, LINK, MESSAGE*} |
| Input | CSDG(*N*, *E*)<br>$N = CL \cup AT \cup OP$,<br>$E = E_{mem} \cup E_{ass} \cup E_{agg} \cup E_{com} \cup E_{gen} \cup E_{cd} \cup E_{rd} \cup E_{wr} \cup E_{oo} \cup E_{cal}$,<br>$\mu_i$, $\mu_i \in N$, $\mu_i$ is a class, attribute or operation in the given class diagram |
| Output | Slice(N', E')<br>$N'=CL' \cup AT' \cup OP'$,<br>$E'= E_{mem}' \cup E_{ass}' \cup E_{agg}' \cup E_{com}' \cup E_{gen}' \cup E_{cd}' \cup E_{rd}' \cup E_{wr}' \cup E_{oo}' \cup E_{cal}'$ |
| Initialisation | CSDG(N,E)=NULL<br>Work=∅, is the work set of CSDG nodes used for slice computing |
| Phase 1 | CSDG construction<br>CSDG=Procedure constructCSDG($D_{cl}$, $D_{sq}$); |
| Phase 2 | Procedure ComputeSlice(CSDG, $\mu_i$) return Slice(*N'*, *E'*)<br>1. N'=∅;<br>2. E'=∅;<br>3. work={ $\mu_i$ };<br>4. while (work≠ ∅) do |

{*select an element $\mu_j$ and remove it from work*}

5. work=work-{$\mu_j$};

6. mark $\mu_j$;

7. for(every unmarked element $\mu_k$ in N) do

8. if (there exists an edge ($\mu_j$, $\mu_k$) in E) do

9. work=work∪{ $\mu_k$};

10. E'=E' ∪ ($\mu_j$, $\mu_k$);

11. end if

12. switch(type of edge($\mu_j$, $\mu_k$)) do

13. case($RD_{ass}$($\mu_j$, $\mu_k$)): do

14. $E_{ass}$'= $E_{ass}$' ∪ {($\mu_j$, $\mu_k$)};

15. end case;

16. case($RD_{agg}$($\mu_j$, $\mu_k$)):do

17. $E_{agg}$'= $E_{agg}$' ∪ {($\mu_j$, $\mu_k$)};

18. end case;

19. case($RD_{com}$($\mu_j$, $\mu_k$)): do

20. $E_{com}$'= $E_{com}$' ∪ {($\mu_j$, $\mu_k$)};

21. end case;

22. case($RD_{gen}$($\mu_j$, $\mu_k$)):do

23. $E_{gen}$'= $E_{gen}$' ∪ {($\mu_j$, $\mu_k$)};

24. end case;

25. case($DD_{rd}$($\mu_j$, $\mu_k$)): do

26. $E_{rd}$'= $E_{rd}$' ∪ {($\mu_j$, $\mu_k$)};

27. end case;

```
28.          case(DD_wr(μ_j, μ_k)): do
29.              E_wr'= E_wr' ∪ {(μ_j, μ_k)};
30.              end case;
31.          case(DD_oo(μ_j, μ_k)): do
32.              E_oo'= E_oo' ∪ {(μ_j, μ_k)};
33.              end case;
34.          case(CD (μ_j, μ_k)):do
35.              E_cd'= E_cd' ∪ {(μ_j, μ_k)};
36.              end case;
37.          case(DEP_cal(μ_j, μ_k)):do
38.              E_cal'= E_cal' ∪ {(μ_j, μ_k)};
39.              end case;
40.      end switch
41.  end for
42.  switch(type of μ_j) do
43.          case(μ_j is in AT):do
44.              AT'=AT'∪ { μ_j };
45.              if(computeSlice(CSDG, μ_j) has not been
                 computed) do
46.                  (N_j', E_j')=computeslice(CSDG, μ_j);
47.                  E'= E' ∪ E_j';
48.                  N'= N' ∪ N_j';
49.              end if
50.              end case;
```

```
51.          case(μj is in CL): do
52.                CL'=CL' ∪ { μj };
                   {*Compute slice based on the slicing
                   criteria(CSDG, μj)*}
53.                if(computeSlice(CSDG, μj) has not been
                   computed) do
54.                     (Nj', Ej')=computeslice(CSDG, μj);
55.                     E'= E' ∪ Ej';
56.                     N'= N' ∪ Nj';
57.                end if
58.                end case;
59.          case(μj is in OP): do
60.                OP'= OP' ∪ {μj};
                   {*Compute slice based on the slicing
                   criteria(CSDG, μj)*}
61.                if(computeSlice(CSDG, μj) has not been
                   computed) do
62.                     (Nj', Ej')=computeslice(CSDG, μj);
63.                     E'= E' ∪ Ej';
64.                     N'= N' ∪ Nj';
65.                end if
66.                end case;
67.           end swtich
68.      end for
69.  end while
```

| | |
|---|---|
| | 70. return CSDG(N', E') |

List 4-2 CSDG Slicing Algorithm

Suppose given a slicing criteria (CSDG, $CL_A$) for Figure 4-10, the slicing result is shown in Figure 4-11 by using the slicing algorithm in List 4-2.

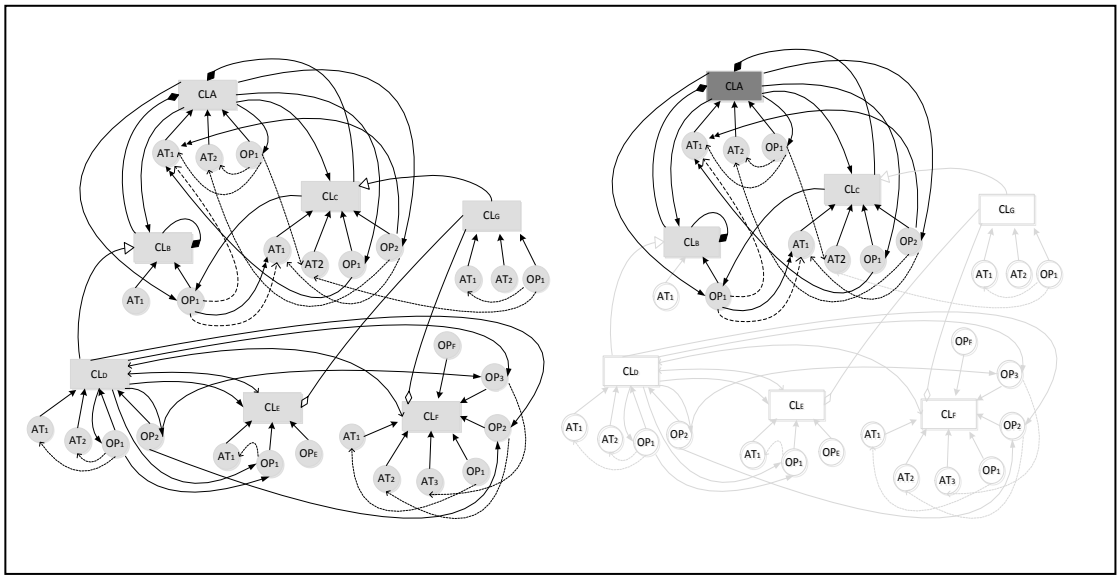



Figure 4-11 Slicing Output using $CL_A$ as the Slicing Criteria

## 4.5 Legacy System Partition

As legacy systems are always huge in size, it is useful to break a complicated system down into a collection of small and manageable subsystems which respectively contains relevant functionalities. Clustering technique is usually used to break down a system into a set of meaningful sub-clusters by using some certain decomposition criteria. High cohesion, low coupling and interface minimisation are the goals that such criteria try to achieve.

Legacy systems always contain a large number of cooperating components, and these components are often organised into identifiable clusters, namely, subsystems. That is, components which contribute to the same business logic are included in a subsystem. Hence, according to domain-expert knowledge, a legacy system is preliminarily decomposed based on the discrepancy of business logic that embedded in various subsystems. Figure 4-12 shows the structure of a legacy system after decomposition.

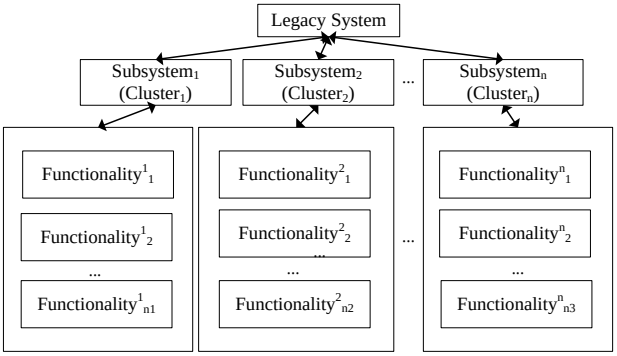



Figure 4-12 Structure of a Legacy System after Decomposition

Based on the CSDG, the following sections will concentrate on clustering legacy systems according to the high cohesion and low coupling principle. The clustering in this thesis refers to the software systems written in object oriented languages, such as java.

### 4.5.1 Relationship Type Weight

As described in previous section, there are different types of relationships among classes in object oriented systems which play different roles in system architecture. Relationships between classes can tell people something about cohesion and coupling of modules, or about the layers built into the legacy system. For example, class *B* inherits from class *A*, class *B* is hardly independent from class *A* because the methods and attributes of class *B* may be inherit from class *A*. When it comes to system decomposition, if class *A* and class *B* are separated into two different components, reusability of the component where class *B* hosts may be low due to its tight dependence on the component where class *A* hosts. It will make the difference if the relationship between class *A* and Class *B* is message link. Therefore, taking into consideration of relationships among different classes in system decomposition may improve the correctness of decomposition.

For a given object oriented system, CSDG defined in Section 4.4 can be constructed based on the extracted static and dynamic models from the object-oriented system to represent the dependency relationship. Suppose $G = (N, E)$ is a CSDG, for any edge $e \in E$, $W(e) = F(t)$ is defined as a weight function as follows:

$$F(t) = \begin{cases} W_{mem}(t = membership) \\ W_{ass}(t = association) \\ W_{com}(t = composition) \\ W_{gen}(t = generalisation) \\ W_{agg}(t = aggregation) \\ W_{cd}(t = control\ dependency) \\ W_{dd}(t = data\ dependency) \\ W_{cal}(t = call\ dependency) \end{cases} \quad (1)$$

It can be concluded from Formula 1, the weight of each edge in CSDG only depends on its relationship type. There are three types of node in CSDG (class node *CL*, attribute node *AT* and operation node *OP*) which accurately describing to what extent the classes are dependent on each other when analysing the dependency relationships among classes. Obviously, it is meaningless if *AT* nodes and *OP* nodes are separated from *CL* nodes and thereby they should be treated as a whole when considering system decomposition. Therefore, the value of $W_{mem}$ should not be taken into consideration of independency metric calculation defined in Section 4.3.2 and thereby its value is set to 0 for the purpose of uniform.

Based on the coupling degree of class diagram relationships, there exist: generalisation > composition > aggregation > association > dependency. Therefore, the weight value of each edge in CSDG is set according to their corresponding relationship derived from coupling degree and shown in Table 4-3.

Table 4-3 Weight Value for Edge in CSDG

| Weight | Relationship | Value | Remarks |
|---|---|---|---|
| $W_{mem}$ | membership | 0 | Membership link |
| $W_{ass}$ | association | 0.6 | relationship link |
| $W_{agg}$ | aggregation | 0.7 | relationship link |
| $W_{com}$ | composition | 0.8 | relationship link |
| $W_{gen}$ | generalisation | 0.9 | relationship link |
| $W_{cd}$ | control dependency | 0.5 | message link |
| $W_{dd}$ | data dependency | 0.5 | message link |
| $W_{cal}$ | call dependency | 0.5 | message link |

### 4.5.2 System Decomposition Algorithm

In this thesis, a greedy algorithm based on [105] is proposed to search for high independent clusters within limited scope of each node taking relationships among different classes into consideration. In [123], suppose that $C = (G_1, G_2, \ldots G_n)$ is a cluster for a given graph $G = (V, E)$, where $G_i = (V_i, E_i)$ $(1 \leq i \leq n)$ is a subgraph of $G$, the quality measurement model for object oriented system decomposition QMOOD is defined as:

$$MQ(C,G) = \frac{\sum_{i=1}^{n} s(G_i,G_i)}{n} - \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} s(G_i,G_j)}{n(n-1)/2} \qquad (2)$$

Where $s(G_i, G_i)$ represents the cohesiveness of $G_i$, and $s(G_i,G_j)$ represents the coupling between $G_i$ and $G_j$. It is obvious to figure out that the higher cohesion of the subgraph, the better the decomposition quality and the higher coupling among subgraph, the worse the decomposition quality. The term subgraph is used in this thesis to represent the cluster in graph domain.

Inspired by QMOOD, for a given $G = (V, E)$ representing an object oriented system with each $V$ representing each class and $E=\{<u, v, t> | <u, v, t> \in E \text{ and } u,v \in V, t \in T\}$, T={membership, association, aggregation, composition, generalisation, control dependency, data dependency, call dependency} is the set of relationships among classes, a sub-graph $G' = (V', E')$ of $G = (V, E)$, the relation set from $G'$ *to its* complementary graph is $O = \{<u, v, t> | <u, v, t> \in E \text{ and } u \in V', v \in V\text{-}V', t \in T\}$, the independent metric (*IM*) defined in [105] is listed in Formula 3 which will be used to calculate the independence for each cluster in this thesis.

$$IM(G') = \frac{\sum_{e \in E'} W(e) - \sum_{e \in O} W(e)}{|V'|} \qquad (3)$$

In Formula 3, three considerations are taken into account including cohesion within subgraph, coupling between subgraph and its complement, the size of subgraph. Cohesion can be obtained by calculating the sum of weights of inner relationships within the subgraph, while coupling is achieved by calculating the sum of weights of outward relationship, and size of subgraph $V'$ can be represented using the number of vertices in the subgraph.

Large software systems are usually composed of basic clusters and composite clusters due to the different abstraction level. Composite clusters can be further decomposed into basic clusters. The proposed decomposition algorithm is composed of two phases. In first phase *IM* in Formula 3 is used to compute the independency metric so that the basic clusters can be identified and then removed from the graph. The remaining of the graph is treated as composite cluster on which further decomposition can be performed in second phase. Detailed description of the proposed decomposition algorithm is specified in List 4-3.

**System Decomposition Algorithm**

| | |
|---|---|
| Input | CSDG *G(N, E)*, representing an object oriented system |
| Output: | The set of candidate clusters *ClusterList*={*CL*'} |
| Initialisation | *ClusterList* = ∅;<br>*Slice(N',E')*=∅, is the slicing output used as intermediate work set of cluster computation;<br>*NodeSet*=∅, is the CL node set in Slice. |
| Phase 1: | Cluster Detection:<br>procedure ComputeCluster(G) return ClusterList<br>1: ClusterList = NULL;<br>2: for each node $N_i$ in graph *G* do<br>3: Slice(*N'*,*E'*)=ComputeSlice(*G*, $N_i$);<br>{*Compute the slice based on the slicing criteria $N_i$ using the slicing algorithm proposed in Section 4.4.2*}<br>4: NodeSet={CL'}; {* Pick out CL node in the slice*}<br>5: *IM*=calculateIM(NodeSet);<br>{*Compute IM according to independence metrics formula*} |

| | |
|---|---|
| | 6: *ContrIMList*=calculateElementContrIM(NodeSet);<br>{*Calculate the contribution value measured by IM of each Node in the NodeSet to itself*}<br>7: TmpNodeSet=DeleteNegativeElements(ContrIMList,NodeSet);<br>{*Delete nodes whose IM value is negative from the NodeSet*}<br>8: *TmpIM*=calculateIM(TmpNodeSet)<br>9: if IM>TmpIM do<br>10: ClusterList.add(NodeSet);<br>11: else<br>12: ClusterList.add(TmpNodeSet);<br>13: end if<br>14: return ClusterList;<br>15: end for |
| Phase 2: | Cluster sorting and removal:<br>After detecting cluster, the cluster list is sorted by their IM values and clusters whose IM values are higher than the threshold are chosen as candidate clusters and removed from the graph. |

List 4-3 System Decomposition Algorithm

The proposed algorithm improves the accuracy of that in [105] in the following reasons:

- The graph used in the decomposition algorithms in [105] is static diagram, while the graph used in this thesis is a combination of static and dynamic diagram. Dynamic diagram depicts the interactivity between classes which cannot be shown in static diagram, and improves the accuracy when evaluating the independency among classes.
- The initial node set of a cluster is obtained by setting the minimal path length as 4 in the algorithm in [105], while it is obtained by model slicing according to the

dependency extent among classes in the proposed algorithm in this thesis which is more accurate than that in [105].

An example in Section 4.3 is used to illustrate the application of the proposed decomposition method, shown in Table 4-4, Table 4-5 and

Table 4-6.

Table 4-4 Results of One Iteration

| Cluster No. | Slice Criteria | Slice Note Set | IM |
|---|---|---|---|
| 1 | $CL_A$ | $CL_A$ ,$CL_B$ ,$CL_C$ | 2.7 |
| | $CL_B$ | $CL_A$ ,$CL_B$ ,$CL_C$ | 2.7 |
| | $CL_C$ | $CL_A$ ,$CL_B$ ,$CL_C$ | 2.7 |
| 2 | $CL_D$ | $CL_A$,$CL_B$,$CL_C$, $CL_D$,$CL_E$ ,$CL_F$ | 2.28 |
| 3 | $CL_E$ | $CL_E$ | 0 |
| 4 | $CL_F$ | $CL_F$ | 0 |
| 5 | $CL_G$ | $CL_A$,$CL_B$,$CL_C$,$CL_E$,$CL_F$,$CL_G$ | 2.15 |

After removal the cluster 1 including the node of $CL_A$, $CL_B$ and $CL_C$ with the highest IM value, the remaining graph is treated as composite and the second iteration decomposition is performed on it. The result is shown in Table 4-5.

Table 4-5 Result of Second Iteration

| Cluster No. | Slice Criteria | Slice NoteSet | IM |
|---|---|---|---|
| 2 | $CL_D$ | $CL_D$ ,$CL_E$ ,$CL_F$ | 1.57 |
| 3 | $CL_E$ | $CL_E$ | 0 |
| 4 | $CL_F$ | $CL_F$ | 0 |
| 5 | $CL_G$ | $CL_E$ ,$CL_F$,$CL_G$ | 1.13 |

Table 4-6 Result of Final Cluster

| Cluster No. | Class name |
|---|---|
| 1 | $CL_A$ ,$CL_B$ ,$CL_C$ |
| 2 | $CL_D$ ,$CL_E$ ,$CL_F$ |
| 3 | $CL_G$ |

## 4.6 Security Mechanisms Detection

The security mechanisms detection process is to identify the existing security countermeasures used in the legacy system so as to conduct a fully evaluation towards security level of the target system. The best way to fulfil this task is to go through the system design specifications and inspect all of software documents from various phases of the development process. However, many legacy systems are lack of such documents due to a variety of reasons, while as the one who knows the system best, system administrator may have some useful knowledge about countermeasures of legacy systems. Under such circumstances, security countermeasure detection can be smoothly performed with the participation of system administrators.

In this section, a method to conduct security countermeasures detection for legacy system, shown in Figure 4-13, is proposed which is divided into two main parts: extraction phase and identification phase. The first phase is automated with a tool, while the other, where the actual analysis takes place, is performed by a human expert who has expert knowledge in security domain.

In the extraction phase, reverse engineering tools are used to extract and capture all relevant information from the legacy system under evaluation and store it in a collection of so-called system model.

Subsequently, in the identification phase, each of the gathered system model is inspected in order to determine whether it is relevant with the security artefacts listed in

security artefacts base which stores a checklist of possible security issues, and is created and maintained by security expert. The results of the identification phase are a mapping list showing whether or not there are any security artefacts in the legacy system and what types of security countermeasures they belong to. It is motivated by searching for security core structure in the software design to determine the built-in security mechanisms of a legacy system.

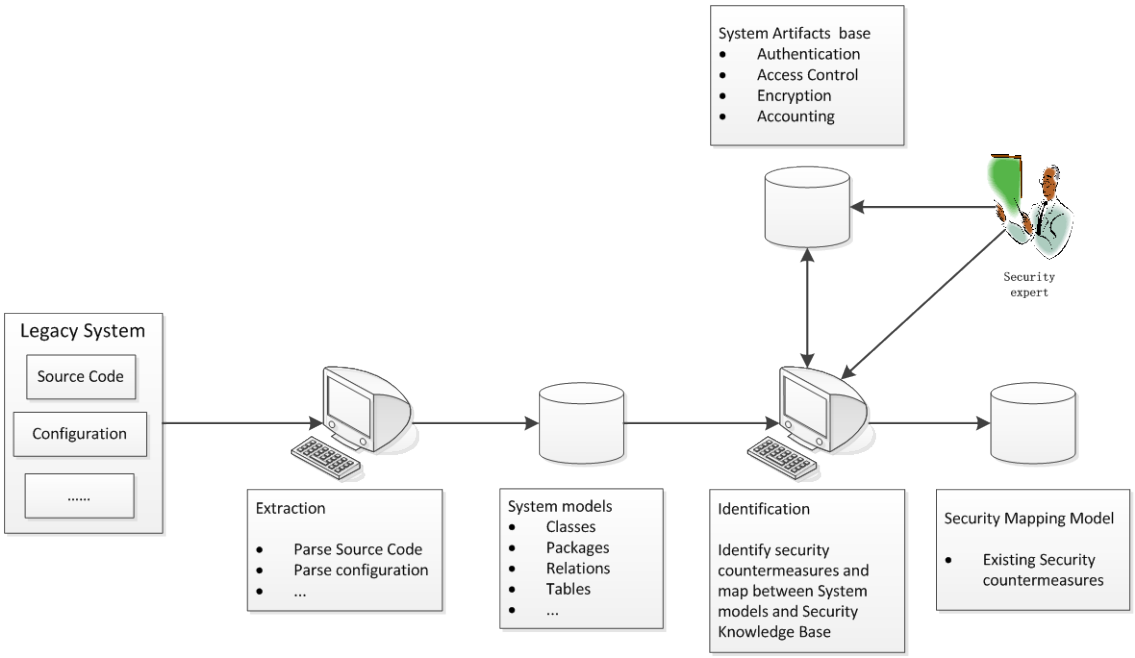



Figure 4-13 Security Countermeasure Detection Overview

### 4.6.1 System Artefacts Extraction

The purpose of the extraction phase is to create a model of the analysed system that stores all basic information necessary for further security mechanism identification. It is constructed using reverse engineering techniques. The result of this phase contains information about diverse software artefacts such as classes, packages, relations between classes or packages, and any other structural information that may potentially contribute to further security analysis. The result of class diagram reverse engineering in Section 4.2 can be used as system artefacts.

### 4.6.2 Security Artefacts Identification

The idea of the identification phase is to detect the security mechanism in use in the

legacy system. It is not easy to detect the built-in security mechanism in a legacy system. For simplification, the process is divided into several steps shown in Figure 4-14.

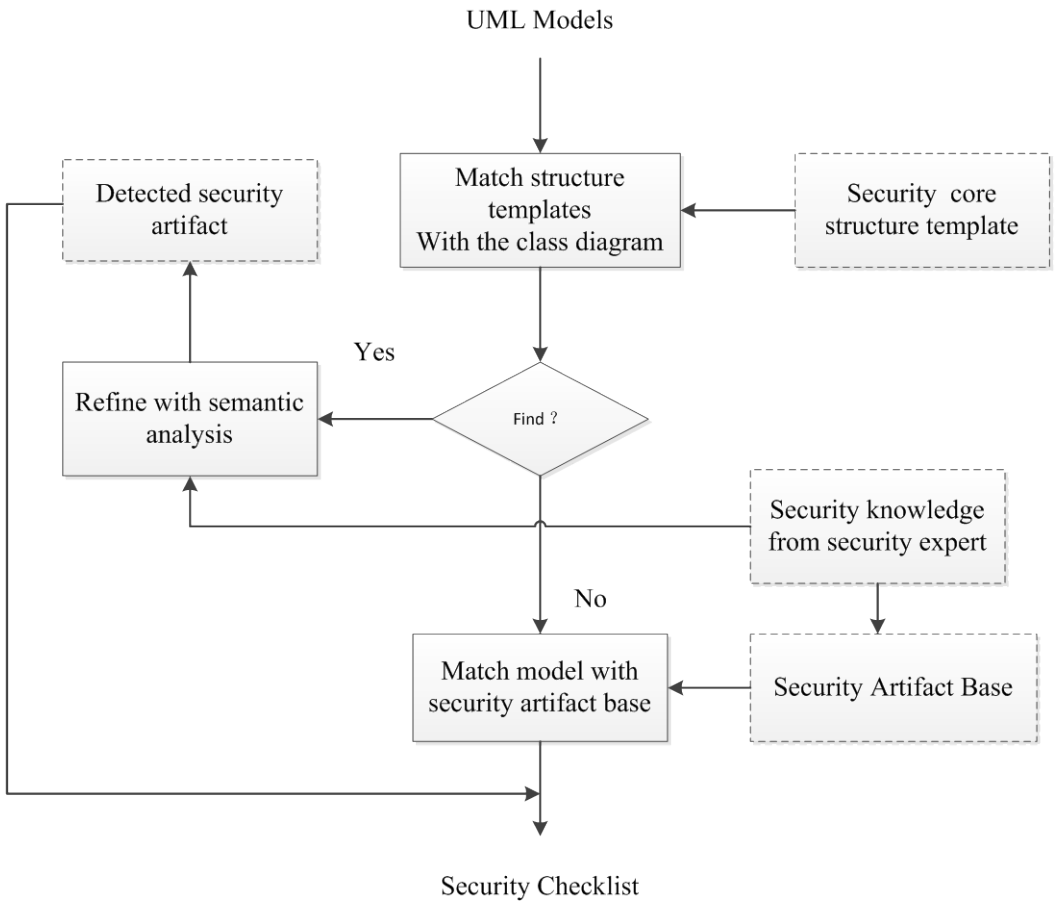



Figure 4-14 Process of Security Mechanism Detection

Firstly, core structure detection is used to match the structure of legacy system with the core security structures, which attempts to find a set of classes whose relationships among them match the relationships among the classes in the core structure of Security Artefacts Base. The core security structure can be handled as a graph with the nodes representing the classes and edges representing the relationships among classes. The proposed method of identification security structure is similar to design pattern detection. However, security core structures derived from security patterns tend to be more abstract and allow more variability in the implementation than design patterns which results in more uncertainty matching [167]. For example, the reference monitor in the authorisation pattern could be a proxy or a wrapper, or some other form of mediator.

In order to eliminate negative matching, it is necessary to make a semantic analysis to the matching structure with the knowledge from security expert. Security patterns can be detected from refinement. Due to the large size and wide range of security patterns, only two most important security patterns, Authentication and Authorisation, will be considered in this thesis.

Although some security concerns such as authentication and authorisation, have several forms of patterns for different needs. But their core elements are common. Recognising the core elements and then building a bigger match could facilitate matching over a range of related patterns [167].

Secondly, on the consideration that the outcome of the first step may be nothing since none of security core structure is detected, security mechanism detection is extended by matching system models with the artefacts listed in security artefact base which is established with the help of security architect or expert. Security checklist will be produced during the two-step identification.

### 4.6.3 Security Artefacts Base

One of the important steps of this method is to establish the Security Artefacts Base. From best practice and ISO 27001 control, it can be concluded that security must embody some form of activities such as authentication, authorisation, encryption and accounting. Security services such as access control, emphasises on a prevention approach to security. Accounting as a detection security approach should also be taken into account in the proposed framework. Other security services such as authentication, supports both protection and detection.

#### 4.6.3.1 Authentication

In authentication, a subject is some kind of user who has to identify itself to the system so as to gain authorisation. There are various ways to make authentication. The typical ones involve password which may possibly use a security service like Lightweight Directory Access Protocol (LDAP) or Radius. Some other mechanisms like ID cards or biometrics are popular authentication means. Figure 4-15 shows the core structure of authentication mechanism.

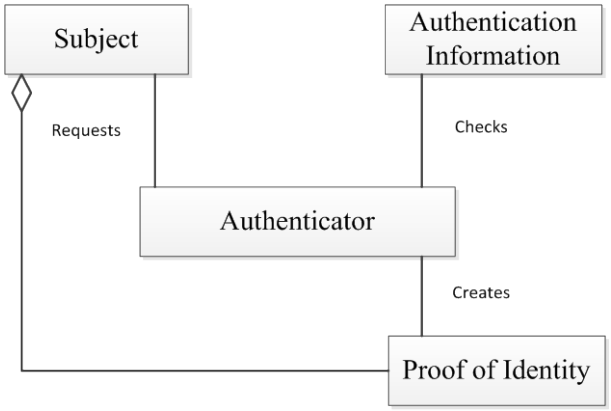



Figure 4-15 Authentication Core Structure

Typical mechanisms used for authentication are listed as follows:

- User ID and Password

  This technique generally scores high on cost effectiveness and usage requirements, but lower on reliability and protection of passwords. Password's ability to avoid confirming imposters is medium at best, because passwords can be obtained through theft or other means. This ability depends on good password practice—for example, the use of hard-to-guess passwords, and not recording passwords in easy-to-find locations. Password's ability to avoid denying legitimate users depends on the likelihood of remembering passwords: good passwords can be somewhat difficult to remember.

- PKI (Public Key Infrastructure)

  This technique depends somewhat on the population to which it is applied. It can score very highly on reliability with a relatively sophisticated user base, but has high cost.

- SSL or password digests

  Basic authentication uses SSL or password digest to protect the password.

- Server-side authentication

  Authentication should be in the server side (instead of client side/JavaScript).

- Credentials

  Authentication token / password are stored with encryption / salted hash.

- SSO (Single Sign On) / centralised security service

  Users don't have to have many accounts/passwords, users don't have to share their passwords with many applications/resources, and the developers don't have to maintain multiple authentication mechanisms in different systems. With federated identity provider, you can centralise the credentials across organisations.

#### 4.6.3.2 Authorisation and Access Control

Access control exists to ensure that unauthorised access is not allowed and that authorised users cannot make improper modifications. A reference monitor is used in access control to check whether or not the access request from the subject to the protected object is authorised. The request with a valid authorisation can be forwarded. In the implementation phase, the reference monitor can be realised as proxy, wrapper or mediator. Figure 4-16 shows the core structure of access control mechanism.

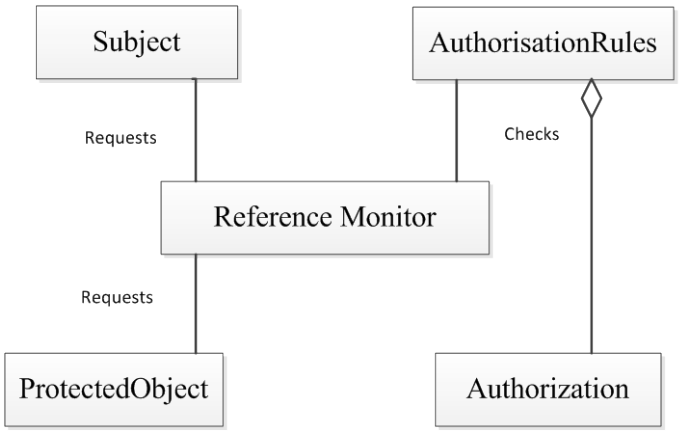



Figure 4-16 Access Control Core Structure

Access control policies are a collection of access control rules. As shown in Figure 4-17, access control rule is a tuple involving entities of Subject, ProtectedObject, and Right which means for a given Subject whose accessible objects are determined by the Right representing by “allow” or “deny”. In implementation phase, the rules may be implemented and stored in a XML document or a table rather than appearing as a class. The subject in access control policies may be a role with related users or with groups and sessions.

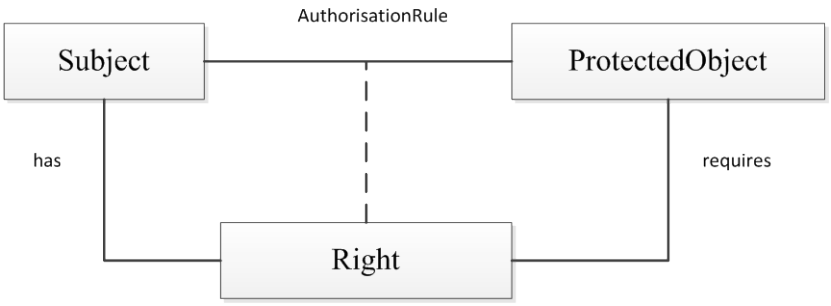



Figure 4-17 Access Policy Core Structure

The typical access control techniques may include：

- Group grants
- Role grants
- Delegation
- Access control lists
- Policy objects
- Rules

### 4.6.3.3 Encryption Mechanism

Encryption techniques play a vital role in secure software from unauthorised access. The well encrypted data or information cannot be read by unauthorised parties even if it is hacked. Encryption is typically required for services involving personal information, for example, online banking.

In an encryption scheme, information or data referred to as plaintext is encoded into a different unreadable form using some kind of encryption algorithm, and the encrypted form is called cipher text. There are various encryption algorithms exist which are open to public. An encryption key is needed to perform the encryption which specifies how the data is to be encoded. An unauthorised party without the key should not be able to understand anything from the cipher text while an authorised one can decode the cipher text with a secret decryption key. The keys are usually produced by a key generation algorithm randomly for security reasons.

Encryption scheme can be classified into two kinds according to the different encryption key schemes:

- Symmetric key encryption
  In such scheme, the key used to encrypt and decrypt are same. Therefore, the communication parties have to reach an agreement on the secret key before starting the communication. The example of symmetric algorithm includes DES, AES, and IDEA etc.

- Asymmetric key encryption
  The keys used to encrypt and decrypt are different. The encryption key also called public key is published to anyone who wish to encrypt the messages, while only the receiving party holds the decryption key and then cipher text can be decrypted and read. The most frequently used asymmetric encryption scheme is RSA.

Key size or key length is the size measured by bits of the key used in an encryption algorithm. Generally speaking, the more bits the key size is, the more strength the encryption algorithm is and the more security of the encryption scheme provides.

- Well-proven encryption algorithms
  Use well-proven encryption algorithms (e.g. AES) in well-proven libraries instead of inventing and implementation your own algorithm.

- Sensitive data encryption
  Encrypt/hash sensitive data e.g. bank-accounts in the LDAP production copy used for development/test.

- Enough encryption key
  Use enough key size. Securely distribute, manage, store the keys, and change the keys periodically.

The equivalent relation between symmetric key and asymmetric key scheme can be found in [131], shown in Table 4-7. RSA security claims that the keys with 1024 bits are likely to become crackable during the time between 2006 and 2010 and that keys with 2048 bits are sufficient until 2030. An RSA key length of 3072 bits should be used

beyond 2030. NIST key management guidelines [131] further suggest that 15360-bit RSA keys are equivalent in strength to 256-bit symmetric keys.

Table 4-7 Comparable Strength between Symmetric Key and Asymmetric Key

| RSA Keys (bits) | Symmetric Keys (bits) |
|---|---|
| 1024 | 80 |
| 2048 | 112 |
| 3072 | 128 |
| 15360 | 256 |

#### 4.6.3.4 Accounting

The function of security accounting is to track security-related actions or events, such as damage to property, attempts at unauthorised database access, or transmission of a computer virus, and provide information about those events [151]. The information provided includes identifying those who participated in the events, so that they may be held accountable. The primary security property supported by security accounting is accountability.

For application domain, security accounting mechanisms are usually some kind of auditing. Audit is the review of events stored in logs, sometimes called audit logs or audit trails, to determine inadequacies during operation or non-compliance with policy. Audit specifically scrutinises information for security relevance. Some form of auditing from [149] is listed below.

- Centralised audit log
- Encrypted checksums on log records
- Encrypted log records
- Digital signature (non-repudiation)

#### 4.6.3.5 Input Validation

Improper or lack of user input data validation is one of the biggest security issues in web applications. The issues include Buffer Overflows, SQL Injection, Cross-Site Scripting attacks, and many others. Input validation is the process to test the correctness of any input to the applications. All application have the specific requirement for user input which could come from various sources, for example, end user, malicious user, another application or any other sources. All input should be validated before processed by application.

Input validation only on the client side is not sufficient in many cases. For example, JavaScript can be used to validate client side input. However, this kind of validation can be easily bypassed by using some proxy tools, such as TamperIE or WebScarab tool. As a result, it is very important to have a validation on server side while make it on the client side at the same time because there is no guarantee that validations on the client will be executed.

#### 4.6.3.6 Session Management

A common vulnerability of web applications is caused by not protecting account credentials and session tokens. There are four types of session id attacks: interception, prediction, brute-force, and fixation. In each attack, an unauthorised user can hijack a session and assume the valid user's identity. Encrypting sessions is effective against interception; randomly assigned session ids protect against prediction; long key spaces render brute force attack less successful; forcing assignment and frequent regeneration of session ids make fixation less problematic. Some security suggestions on session management are listed as follows.

- HMAC(Hash Message Authentication Code) or encrypt session ID
- Logout mechanism
- One time Nonce
- Random generated Session ids
- Session ID length 128 bits or more

#### 4.6.3.7 Configuration Management

Configuration management also referred to as change management, it documents much configuration information in the system, for example, how the networks devices are configured, what version of the application are running, and what are the last patches installed on the devices. There is no doubt that attackers are interested in this information. Therefore, how to manage configurations has close relations with system security. The following lists typical suggestions of configuration management for security concerns.

- Restrict access to configuration file
- Encrypt/hash sensitive configuration data
- Centralised security management
- Configuration change detection
- Restrict message size

#### 4.6.3.8 Error Handling

Error is inevitable during software development. Errors may force an application stop working. How to solve the error securely is vital to application security. Detailed error information may include sensitive information which can be exploited by attacker. Therefore, error messages should be handled properly so as to avoid being exploited by attackers. The typical error handling suggestions are listed as follows.

- Policy for handling errors
- Hide sensitive information in the error pages
- Centralised error handling

After identifying the system artefacts with security artefacts base, an assessment checklist is to be generated which has three columns in total. The first two columns enumerate the security services and the security mechanisms supporting the services. The next one is checkbox denoting if the mechanism is implemented in the current design.

### 4.6.4 An Example

Let's consider an application architecture that delivers a Web-based business-to-consumer portal that integrates a variety of back-end applications. The application security architecture adopts a basic authentication using username and password for authenticating the user, authorises the user as a customer or administrator to perform further operations, and captures all events and actions using a logging mechanism for accountability. The back-end applications running on heterogeneous platforms make use of a shared security context to provide single sign-on access and to participate in portal-initiated transactions. Table 4-8 shows the checking result of the given example.

Table 4-8 Example of Security Implementation Checklist

| Security Service | Security Mechanism | Current |
|---|---|---|
| Authentication | Username and password | √ |
| | Client-certificate | √ |
| | SSL or Password digest | |
| | Credentials | |
| | PKI | |
| | Server-side Authentication | |
| | SSO / centralised security service | √ |
| Authorisation and Access Control | Group grants | √ |
| | Role grants | |
| | Delegation | |
| | Access control lists | |
| | Policy objects | |
| | Rules | |
| Encryption | Well-proven encryption algorithms | √ |
| | Sensitive data encryption | |
| | Symmetric encryption key 80 bit or more | |
| | Asymmetric encryption key 1024 bit or more | √ |
| Accounting | Centralised audit log | √ |
| | Encrypted checksums on log records | |
| | Encrypted log records | |

| | | |
|---|---|---|
| | Digital signature (non-repudiation) | |
| Input Validation | Server-side validation | |
| | Client-side validation | |
| Session Management | HMAC or encrypt session ID | |
| | Logout mechanism | |
| | One time Nonce | |
| | Random generated session ID | |
| | Session ID length 128 bits or more | |
| Configuration Management | Restrict access to configuration file | |
| | Encrypt/hash sensitive configuration data | |
| | Centralised security management | |
| | Configuration change detection | |
| | Restrict message size | |
| Error Handling | Policy for handling errors | |
| | Hide sensitive information in the error pages | |
| | Centralised error handling | |

## 4.7 Summary

The analysis and extraction of a legacy system are very important before the security oriented evolution. There are huge amounts of legacy software applications which must be understood to decide whether they are needed to be evolved under the specific environment and security requirements. In order to analyse and understand the legacy system at model level, an intermediate representation graph is defined with the consideration of different relationships among classes and dynamic scenario information. The contents covered in this chapter are concluded as follows:

- Firstly, class diagram and sequence diagram of UML models are chosen to be the stereotype models and extracted from the legacy system using existing reverse engineering tools. The process is the basis for the subsequent steps and as an initial step for the evolution of legacy systems.
- Secondly, dependency relationships among classes in static and dynamic models are analysed and an intermediate representation graph called CSDG is constructed

based on which a model slicing method is proposed to assist the legacy system understanding and security analysis in the later phase.

- Thirdly, software clustering technique is adopted to decompose the legacy system into several clusters. An improved decomposition algorithm is introduced to extract the independent components based on the proposed CSDG graph using the independence metric. The architect's view and users' requirements help to decide the cutting point in the dendrogram. Reverse engineering plays a vital role in the process.
- Fourthly, method of existing security mechanisms detection for legacy systems is presented to help the decision making whether or not the legacy system needs to be evolved by evaluating the risk assessment discussed in Chapter 5. Detection is composed of two steps：known security core structure matching and security artefact base mapping. A security checklist is acquired in this process which is further analysed in Chapter 5.

# Chapter 5

# Security Requirements Elicitation

**Objectives**

- To present a threat modelling method for web application
- To propose a risk assessment framework for security requirement elicitation
- To evaluate whether the provided security features satisfy the user security objectives
- To conclude the security requirements

## 5.1 Overview

In Chapter 4, UML models can be generated by reverse engineering the legacy software. Refining the UML models to achieve security related artefacts shows a clear and focused view for security analyser. A method to detect existing security mechanisms in a legacy system is proposed in Chapter 4 as well. However, environment where the legacy systems are hosted may change with the rapid development of information technology. Before determining and adopting any security design strategies, it is necessary to perform a risk assessment of the application security architecture to decide whether or not the current security designs in use protect against the security risk to the legacy system and to what extent they satisfy the security needs from users. The following section will focus on the risk assessment and security evaluation of legacy system.

The increasing number of attacks on software systems makes it more important than ever to develop secure software systems. Especially web-based applications and services are facing numerous threats due to their public access. As introduced in

Chapter 2, a legacy system is the system that was developed in the past or has become outdated in the business area. Typically, more and more systems become legacy because of the use of modern software engineering methods and the ever changing requirement. Security as an important non-functional requirement has become one of the key driven factors to software evolution. Integration of security into software engineering has not yet been achieved completely as the amount of security knowledge, including theoretic models, technologies and standards, developed until now is complex, often diffused, and seldom structured enough to be used in software development process [34].

Concrete security requirements and their security risks to the software system should be examined. In the following sections of this Chapter, a novel threat elicitation approach [63] and a risk assessment method [62] are proposed which have been published in the international journals.

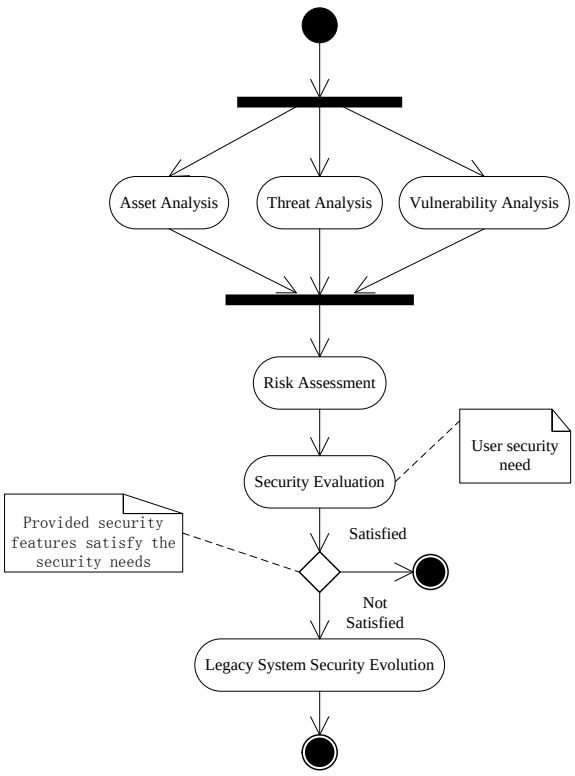



Figure 5-1 Activity Diagram of Risk Assessment and Security Evaluation

Due to the complexity of security implementation, legacy systems are developed using different kinds of techniques and deployed diversity of security countermeasures. Therefore, it is essential to make an assessment to the security risks that the legacy

systems face, whether or not it deploys the security countermeasure and to what extent the assets are protected against the threats. During the assessment process in this chapter, many criteria are required to rank the alternatives of the decisions. To decide whether or not the legacy system needs to be evolved for security, an assessment framework is proposed and the assessment process is described in Figure 5-1. The operational framework of Chapter 5 is shown in Figure 5-2. There exist several kinds of models after the processing of the first phase in SEMDA including system design models and higher level of domain models. These models can be further processed to produce security requirement conceptual model.

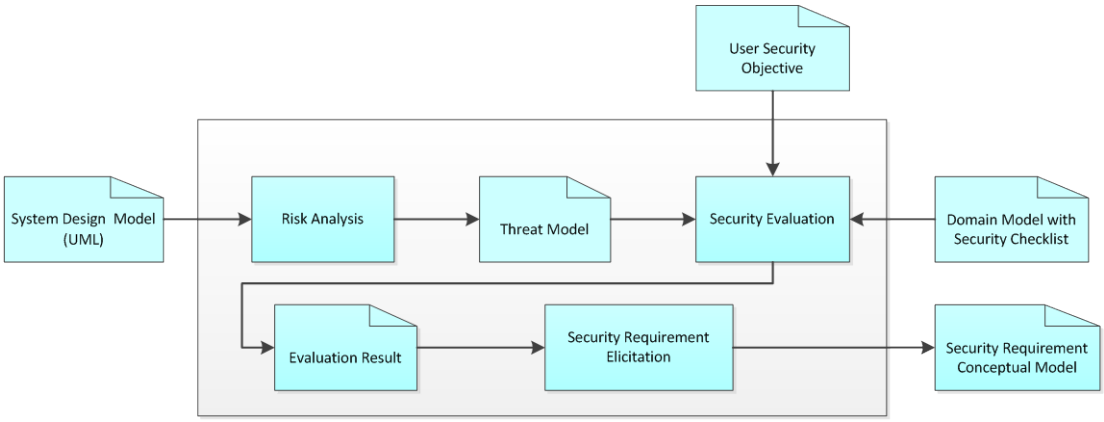



Figure 5-2 Operational Framework for Chapter 5

## 5.2 Risk Assessment

Risk assessment is a process of assessing the security level in a quantitative or qualitative mode of an organisation by evaluating one's exposure to the threats to its assets and operating capabilities.

There exist different kinds of application software. Due to the universal adoption of web, web based applications tend to increasing rapidly and especially popular on mobile devices such as smart phones and tablets. Distinctions between web-based applications written in HTML, JSP or any other web technologies using the web browser and traditional applications written with whatever programming languages running in the traditional computers have emerged. There also exist some debate on whether or not web based application should replace traditional applications. Anyway, web

applications have indeed greatly increased in popularity for many uses.

Hence, web applications are chosen to be the research case due to their popularity in use and complex security concerns. Risk assessment for web application is one of the effective methods to help decision makers determining how much they need to invest in security so as to achieve a desired security level. It is the process to make sure all risks have been considered and thereby make it possible to find proper countermeasures to mitigate those risks. Threat risk modelling methods can be used to facilitate this process.

As described in Section 2.6, several important concepts are usually used in risk analysis, including Asset, Threat, Vulnerability, Risk and Attack. Figure 5-3 shows how different security factors, involved in risk analysis, are related. It is obvious that the target of attack is the assets whose vulnerabilities are exploited by threats which in turn lead to risks and do harm to assets.

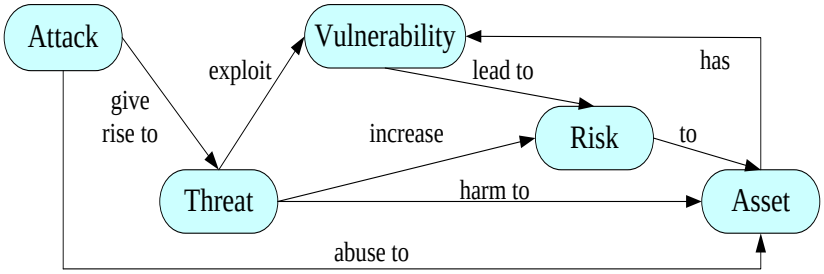



Figure 5-3 Security Concepts Relationship

In this section, a quantitative risk assessment method for web application taking consideration of criticality of asset, threat and vulnerability is shown in Figure 5-4.

The proposed model consists of five phases with difference phases in different colours. Every phase in the framework needs more than one step. Here are the detailed meanings of each phase.

**Phase 1: Architecture and Environment Analysis**

- **Step 1:** Architecture Analysis. System architecture analysis is the recognition process of entire system architecture and business processes so as to precisely understand the platform structure, security boundary, business processes, internal and external environment of the target system. The system architecture

model can be established which is the foundation for data flow analysis.

- **Step 2:** Application Type Identification. In order to have a precise risk estimation of the target web application, it needs to be classified into one of types (from web-app1 to web-app6) according to the proposed web application classification in Section 5.2.2.1. When it is done, the security risks of this kind of web application can be evaluated preliminarily.

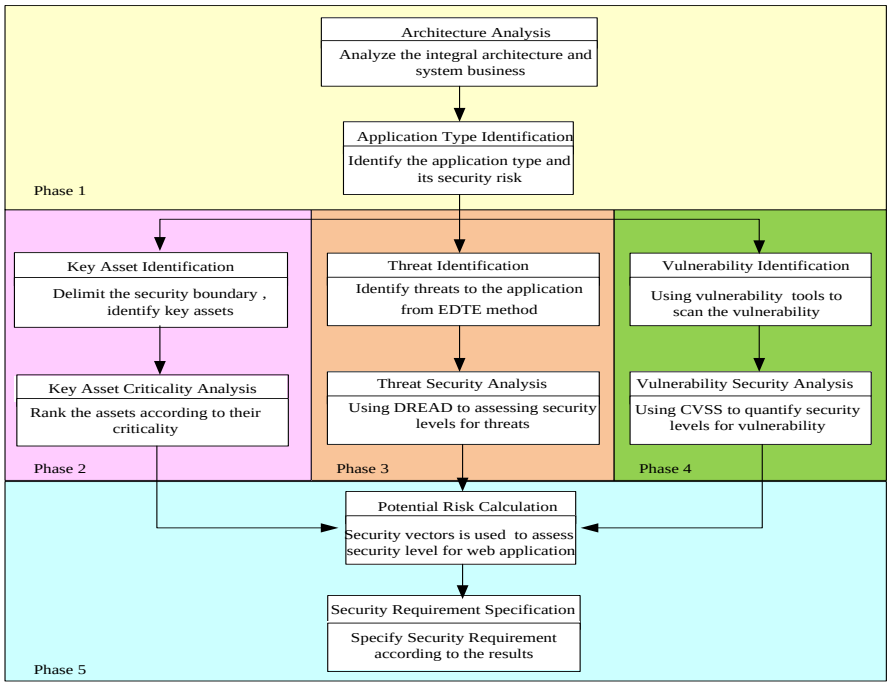



Figure 5-4 Proposed Risk Analysis Process

**Phase 2: Key Asset Analysis**

- **Step 1:** Key Assets Identification. In this step, the key information assets representing as the key data and services which determine system security should be identified. The key information assets are the kernel for risk assessment.
- **Step 2:** Asset Criticality Ranking. The criticality of assets is determined based on the type of information handled in the applications.

**Phase 3: Threat Analysis**

- **Step 1:** Threats Identification. Threat identification is the process of recognising the threats related to each key asset identified in phase 2. The result of threat

identification is a list of threats associated with the target system.

- **Step 2:** Threat Quantification. Threat quantification is an important step to risk assessment and it is the process of quantifying the threats listed in the previous step by using threat DREAD [139] which is a risk assessment model proposed by Microsoft. A risk value may be calculated to each threat after this step.

**Phase 4: Vulnerability Analysis**

- **Step 1:** Vulnerability Identification. The diagnosis tools can be used to perform vulnerability analysis. A list of system vulnerabilities can be detected after this step.
- **Step 2:** Vulnerability Security Scoring. In this study, vulnerability "scoring" systems - Common Vulnerability Scoring System (CVSS) [116] is used to quantify each vulnerability that identified from the previous step, and produce a total score for the vulnerabilities of system.

**Phase 5: Risk Analysis and Security Requirement Specification**

- **Step 1:** Potential Risk will be evaluated based on the results of asset, threat and vulnerability analyses using the security vector <*A, T, V*> in Formula (4).
- **Step 2:** Security Requirement Specification. Security requirements can be specified according to the results of the previous phases.

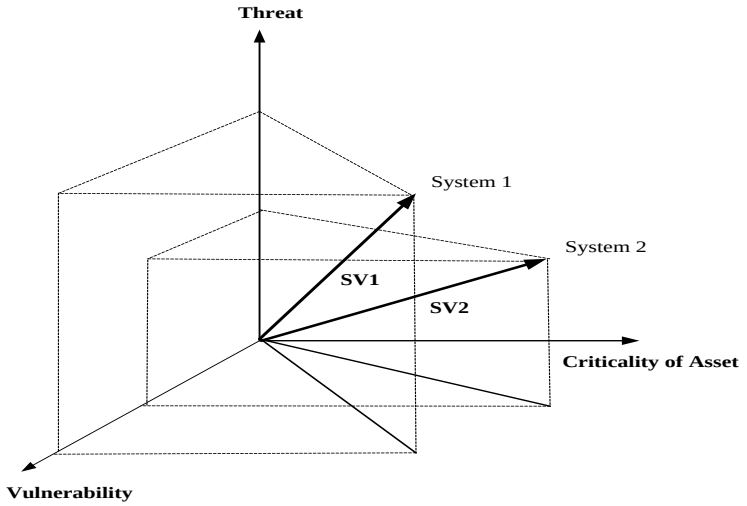



Figure 5-5 Security Vectors

From Figure 5-5, it can be concluded that the computation of security evaluation vectors is the combination of the factors themselves (*A* for Asset, *T* for Threat, *V* for Vulnerability) and the weight of each factor.

$$\mathrm{SV}=\sqrt{A^2 \times W_a + T^2 \times W_t + V^2 \times W_v}\,. \tag{4}$$

Where $W_a$ for the weight of asset, $W_t$ for the weight of threat and $W_v$ for weight of vulnerability. The values of $W_a$, $W_t$ and $W_v$ range from 0 to 1 and it must meet the restriction that the sum of them equals to 1. In this case, taking consideration of importance of each factors, the weights of them are treated as the same which can be formulised as

$$\mathrm{SV}=\sqrt{\left(A^2 + T^2 + V^2\right)/3} \tag{5}$$

### 5.2.1 Asset Analysis

Asset is the protection object in an information system. Asset analysis is the process to discover all assets relevant to the legacy system and rate the criticalities of each of them in order to prioritise them with respect to information security.

In this section, an environment critical asset assessment method is proposed to perform the asset identification and asset criticality analysis. The asset analysis process is briefly explained based on the international standard ISO 17799, improved by BS7799 [19]. The proposed method to assess the assets in information system includes analysing information security requirements, understanding asset criticality, and checking sensitivity of data asset.

#### 5.2.1.1 Asset Identification

The British BS7799 [19] suggests the asset classification as follows:

- Information Asset: DB, data file, system document, user manual, study and training materials, regulations for management, plan document, provision for alternative system
- Documents: contracts, guidelines, company documents, important business

documents

- Software Asset: applications S/W, system S/W, development tool and utility
- Physical Asset: computer and communication equipment, magnetic tape, magnetic disk, power supply, air conditioner, furniture, facilities
- Personnel Asset: individuals, customer, subscriber
- Image and Reputation of a Company
- Service: computer and communication service, warm, light, air conditioning

This thesis attempts to deal with the assets associated with information system rather than all the listed assets in an organisation. Thus, the following asset domains are concerned in this thesis:

- Software: including applications to support the objectives and business processes in an organisation
- Information: including data to achieve the objectives and business process

Asset identification plays a key role in risk assessment. Systematically and explicitly identify the types of assets that need protection and determine the types of protection they need is essential. This activity is typically performed by a system architect or strategic planner, and includes following steps:

- Identify the asset type, as described in this section, the following assets are concerned in this thesis
  - Information or data assets, such as personal and financial data
  - Software asset, such as applications to support the business processes
- Identify sensitive business processes
  - Business process, such as ordering processes
  - Sensitive process, such as logging process, paying process etc.
- Identify what types of security may be needed for each asset type
  - Confidentiality, protection against inadvertent or unauthorised disclosure

- Integrity, protection against inadvertent or unauthorised modification
- Availability, making business assets available for authorised use
- Accountability, attribution of responsibility for action

- Prioritise each assets according to the proposed ranking criteria
  - Sensitivity, such as personal information should be ranked higher criticality
  - Application environment, such as external use should be ranked higher criticality

In this thesis, an asset identification method is proposed by capturing vocabularies of the legacy system with the help of system diagrams. The vocabularies of a system include all the software and conceptual information which are treated as the assets that needs to be identified during the asset analysis phase for the legacy system. System diagrams extracted from the reverse engineering phase can facilitate the identification of information assets. In this section, a diagram based approach is proposed to extract or discover all the assets from system diagrams derived from the legacy system.

For those software designed using traditional techniques, data flow diagram (DFD) illustrates how data is processed by a system, what the system accomplishes and how it will be implemented, when it is refined with further specification. Asset identification for such information-processing systems can be facilitated by analysing DFD. Components of DFD-- data store, flow, and process can be treated as assets and further analysed according to their relevance to sensitive information in the system.

For the system designed using object oriented technique, UML diagrams are commonly used to modelling all aspects relevant to the being developed system in the design phase. UML 2.0 can be classified into 13 diagrams. Among all of the UML diagrams, class diagram shows the structure of the entire system and involves all the possible elements in the system, which is the most appropriate diagram for asset extraction.

When extracting asset from class diagram, all the fields of each class need to be examined by the system designer, security analyst along with the stakeholders of the given legacy system. If the field is thought to be of importance to the organisation or

having impact to system security, it should be treated as an asset. The first field of a class in the class diagram is the class's name which is a good source to find asset. For example, a class with the name of MedicalHistory can be treated as an asset if a hospital management system is considered. The second field of a class in the class diagram lists the attributes of the class which can facilitate the judgement whether a class is a good asset or not. The last section describes the methods or operations of a class which is used to depict the interaction or behaviour of this class to other classes. Operations can be used to convey information based on which the data flows among different objects in the system. Therefore, carefully examining the operation can be another good source to discover asset.

Based on the analysis above, Table 5-1 concludes information asset category and security protection for web application.

Table 5-1 Information Asset Category and Security Protection of Web Application

| Asset types | Type | Security Needed | Description |
|---|---|---|---|
| Internal data | Data | Confidentiality<br>Integrity<br>Accountability | Privacy information may be contained in this data |
| Public data | Data | Integrity<br>Availability | Unauthorised modification or unavailable of this data or could result in loss of business reputation |
| Sensitive data | Data | Confidentiality<br>Integrity<br>Accountability | This kind of data loss or unauthorised modification could result in financial loss |
| Sensitive process | Business process | Confidentiality<br>Availability<br>Accountability | The business processes contain sensitive information processing service. Unauthorised access or unavailable of these service may result in financial loss or business reputation. |
| Non-sensitive process | Business process | Availability | The business process may contain the key services for the customer. Unavailable of this service may affect the business task. |

#### 5.2.1.2 Asset Criticality Quantification

The criticality of assets is treated according to their property and the environment where they host. If the application deals with personal information such as name, password, and finance related information such as credit card and bank account information, and the application is designed for external use, the criticality level is treated as high. Otherwise, the criticality level is treated as medium if the application deals with personal information for internal use, and low if it does not deal with personal information. The relationship of asset criticality with the processed data and its applied environment is shown in Table 5-2.

Table 5-2 Asset Criticality Scale

| Asset Type | Sensitive | Environment | Criticality Level | Risk Scale | Description |
|---|---|---|---|---|---|
| Data<br>Business process | No | Internal use | Very low | 0.0-1.99 | The asset has insignificant importance and has no security requirements |
| | No | External use | Low | 2.0-3.99 | The asset has minor financial value. Comprise of it results in little business impact |
| | Yes | Internal use | Medium | 4.0-5.99 | The asset is of moderate value. It has some security needs and financial value. Comprise of it would impede the enterprise's mission |
| | Yes | External facing known users | High | 6.0-7.99 | The asset is highly value because of its security requirements or customer focus. Its loss would result in considerable harm to customer services and reputation |
| | Yes | External facing public users | Very high | 8.0-10.0 | The asset represents or supports a critical business function. Loss or damage of it results in severe financial or reputation |

### 5.2.2 Threat Analysis

Web application is some kind of different one due to its application environment and complexity. Accordingly, threats to this kind of software are different to some extent. The number and category of the threats to different kind of web applications may not be the same when taking account of complexity and environment where the applications are hosted. Sometimes, the applications may be developed with security in mind, and may be difficult to penetrate as well, but if the environment where the application is hosted is not properly secured, it is easy to penetrate the environment , and as a result, it is easy to compromise the whole application including its subsystems and platform.

As far as the web application developers are concerned, on one hand, they need to keep security on mind when developing the web application, on the other hand, they are usually forced to face the dilemma that how to trade-off among so many product factors such as security requirements, product deadline and budgets etc. Thus, this section proposes a novel approach to ease the elicitation of the threats for web applications by defining web application classification as the filter to rule some threats out immediately according to the security requirement and the given scenery. Before diving into the details of the proposed model, it is better to give an overall idea of this model, described in Figure 5-6, where *SR* represents the security requirement and *Category* represents the web application category.

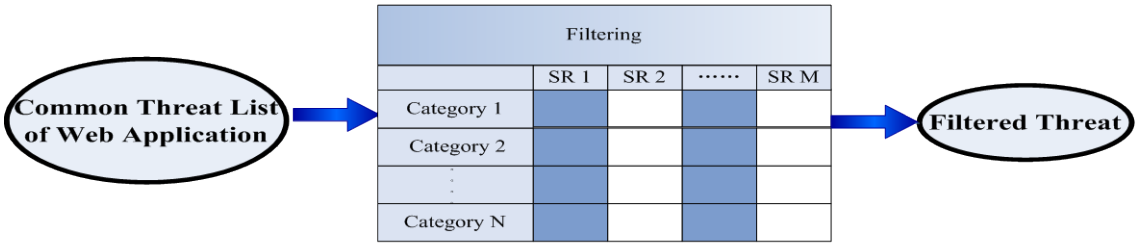



Figure 5-6 Illustration of the Proposed Approach EDTEM

The proposed model, called environment-driven threat elicitation model, short as EDTEM, is working as a sieve to sift the inappropriate threats. With this approach, it starts with a laundry list of common threats [115] grouped by network, host, and application categories. Next, apply the threat list to the given application architecture and screen out the threats matching its own web application category. Then, further

filtering can be done to the result threat set according to the security requirements of the given web application. Some threats will be ruled out because they do not apply to the scenario of the given application. As a result, a set of filtered threats specific to the given web application can be obtained.

This section depicts a new method to ease eliciting threats to web application by using web application classification which is proposed taking consideration of security-related environmental factors.

#### 5.2.2.1 Classification of Web Application

All web-applications are not same, the architecture and its supporting systems could be different for each application depending on the complexity, but the resources or techniques needed for running those applications may be same. So, it is possible to create a common threat model and identify all possible threats that can be used for all web-applications [62]. The Web Application Classification is a cooperative effort to clarify and organise the types of web application with different security risk level so as to ease the process of threats identification. It helps you to identify which threats are relevant to your application through the proposed model.

It is certainly that the risks to web applications are certain to be different when they are hosted in well secured and non-secure environments. Thus, it is necessary to take consideration of the web application type before further discussion.

For the consideration of the environment where web application hosts, three attributes of web applications have been taken into account, which are the usage scope, target user and connectivity mode. Then, the circumstances of all the attributes of web application are listed in Table 5-3. Some symbols are used to represent these attributes, *US* (Internal use only, Internal and External use, External use only) for the usage scope and its values, *TU* (Known users, anonymous) for target user, and *CM* (Intranet, VPN, Internet) for connection mode.

Table 5-3 Attributes of Web Application

| Usage Scope (US) | Target User (TU) | Connection Mode (CM) |
|---|---|---|
| Internal use only (US1) | Know users (TU1) | Intranet (CM1) |
| Blended use (US2) | Anonymous (TU2) | VPN (CM2) |
| External use only (US3) | | Internet (CM3) |

The number of web application types abbreviated as WA equals to:

$$WA = \text{Card}(\text{Domain}(US)) \times \text{Card}(\text{Domain}(TU)) \times \text{Card}(\text{Domain}(CM)) \quad (6)$$

Where Card(Domain(*US*)) denotes the cardinality of Domain(*US*). There should be 18 web application types according to the Formula 6. However, some of them are not applicable in real web application which should be ignored. For example, the combinations of (*US1, TU1, CM3*) or (*US1, TU2, CM3*) are not incompatible. All of the applicable web application types are listed in Table 5-4 with the bigger value the higher possibility of risk level.

Table 5-4 Web Application Classification

| ID | Name | Attribute | Definition | Security Risks |
|---|---|---|---|---|
| WA1 | Internal use facing known users via intranet | US1<br>TU1<br>CM1 | Application used primarily on the internal network of an organisation for a mount of known users. | This kind of applications is designed for internal use so that only internal users can access from intranet. Therefore, the security risk is considered as low. |
| WA2 | Internal blended<br>External use facing known users via VPN | US2<br>TU1<br>CM2 | Application used primarily on the internal network of an organisation, but amount of known external clients can access through VPN | The security risk is low but there are possibilities for sharing user-credentials, impersonation and sniffing on the external client site. |

| | | | | |
|---|---|---|---|---|
| WA3 | External use facing known users via Internet | US3<br>TU1<br>CM3 | Application used for external use. A mount of known users can access from the Internet | The security risk is a bit higher compared to previous types because it is exposed to all kinds of attacks from internet, however, it is not very high for only known users can access |
| WA4 | External use facing public users via Internet | US3<br>TU2<br>CM3 | Application used for external use. Public users can access from internet | The security risks of these applications are considered little bit high compared to previous types since they are open to public from Internet |
| WA5 | Internal blended external use facing known users via Internet | US2<br>TU1<br>CM3 | Application used for internal users and external known users from Internet | The security risks of these applications are higher due to their design complexities. Usually, this kind of applications are designed primarily for internal use, it is a little more dangerous when known users access from Internet |
| WA6 | Internal blended external use facing public users via Internet | US2<br>TU2<br>CM3 | Application used for internal users and external public users from Internet | The security risks of these applications are highest due to their design complexities. Usually, this kind of applications are designed primarily for internal use, it is the most dangerous when public users access from Internet due to lack of security controls |

#### 5.2.2.2 Threat Identification

The methods of threat elicitation mentioned in Section 2.7 are general purpose for all kinds of applications. However, web application is some kind of different one due to its application environment and complexity. Accordingly, threats to this kind of software are different to some extent. The number and category of the threats to different kind of web applications may not be the same when taking account of complexity and environment where the applications are hosted.

In this section, the proposed web application classification is used to filter the threats proposed in [115] according to the given security requirements. In order to illustrate the

detailed the steps of our approach, threats list used as the threats set to be filtered is described below.

For comprehensiveness, a collection of threats proposed by [115] are chosen which enumerates the popular threats that affect web applications at the network, host, and application levels. For the sake of being used conveniently in this thesis, two related factors, web application category and CIAA requirement are added in Table 5-5, Table 5-6, Table 5-7.

Table 5-5 Network Level Threat [115]

| No. | Threat Name and Description | WA Category | CIAA Risk |
|---|---|---|---|
| 1 | **Information Gathering**<br>Information (Network device type, operating system and application versions) may be detected by port scanning in order to perform attack | WA3 WA4<br>WA5 WA6 | C |
| 2 | **Sniffing**<br>Monitoring traffic on the network for data such as plaintext passwords or configuration information | WA3 WA4<br>WA5 WA6 | C |
| 3 | **Spoofing**<br>Spoofing may be used to hide the original source of an attack or to work around network access control lists (ACLs) that are in place to limit host access based on source address rules | WA3 WA4<br>WA5 WA6 | C<br>I |
| 4 | **Session Hijacking**<br>Session hijacking deceives a server or a client into accepting the upstream host as the actual legitimate host. Instead the upstream host is an attacker's host that is manipulating the network so the attacker's host appears to be the desired destination | WA3 WA4<br>WA5 WA6 | C<br>I |
| 5 | **Denial of Service**<br>Denial of service denies legitimate users access to a server or services. The SYN flood attack is a common example of a | WA3 WA4<br>WA5 WA6 | A |

| | | | |
|---|---|---|---|
| | network level denial of service attack | | |

Table 5-6 Host Level Threat [115]

| No. | Threat Name and Description | WA Category | CIAA Risk |
|---|---|---|---|
| 6 | **Viruses, Trojan horses, and Worms**<br>Although these three threats are actually attacks, together they pose a significant threat to web applications, the hosts these applications live on, and the network used to deliver these applications | ALL | C<br>I<br>A |
| 7 | **Footprinting**<br>Examples of Footprinting are port scans, ping sweeps, and NetBIOS enumeration that can be used by attackers to glean valuable system-level information to help prepare for more significant attacks | WA4<br>WA6 | C |
| 8 | **Password Cracking**<br>The attacker cracks the password if the default account names are used. The use of blank or weak passwords makes the attacker's job even easier | ALL | C |
| 9 | **Denial of Service**<br>An attacker can disrupt service by brute force against your application, or an attacker may know of a vulnerability that exists in the service your application is hosted in or in the operating system that runs your server | ALL | A |
| 10 | **Arbitrary Code Execution**<br>If an attacker can execute malicious code on your server, the attacker can either compromise server resources or mount further attacks against downstream systems | WA3 WA4<br>WA5 WA6 | C<br>I |
| 11 | **Unauthorised Access**<br>Inadequate access controls could allow an unauthorised user to access restricted information or perform restricted operations | ALL | C<br>I |

Table 5-7 Application Level Threat by Application Vulnerability Category [115]

| | | | |
|---|---|---|---|
| **Input Validation** | | | |
| 12 | **Buffer Overflow**<br>Buffer Overflow exploits are attacks that alter the flow of an application by overwriting parts of memory | WA3 WA4<br>WA5 WA6 | I |
| 13 | **Cross-Site Scripting (XSS)**<br>An XSS attack can cause arbitrary code to run in a user's browser while the browser is connected to a trusted Web site | WA4<br>WA6 | I |
| 14 | **SQL Injection**<br>A SQL injection attack exploits vulnerabilities in input validation to run arbitrary commands in the database | WA3 WA4<br>WA5 WA6 | I |
| 15 | **Canonicalization**<br>Canonicalization attacks can occur anytime validation is performed on a different form of the input than that which is used for later processing. | WA4<br>WA6 | I |
| **Authentication** | | | |
| 16 | **Network Eavesdropping**<br>An attacker armed with rudimentary network monitoring software on a host on the same network can capture traffic and obtain user names and passwords | WA3 WA4<br>WA5 WA6 | C |
| 17 | **Brute Force Attacks**<br>Brute force attacks rely on computational power to crack hashed passwords or other secrets secured with hashing and encryption | WA4<br>WA6 | C<br>A |
| 18 | **Dictionary Attacks**<br>An attacker uses a program to iterate through all of the words in a dictionary (or multiple dictionaries in different languages) | WA4<br>WA6 | C<br>A |

| | | | |
|---|---|---|---|
| | and computes the hash for each word | | |
| 19 | **Cookie Replay**<br>An attacker captures the user's authentication cookie using monitoring software and replays it to the application to gain access under a false identity | WA3 WA4<br>WA5 WA6 | C |
| 20 | **Credential Theft**<br>Credential theft occurs when an attacker obtains and uses valid account credentials (username and password) for unauthorised access to a computer | ALL | C<br>I |
| **Authorisation** | | | |
| 21 | **Elevation of Privilege**<br>An attacker may try to elevate privileges to a powerful account such as a member of the local administrators group or the local system account | ALL | C<br>I |
| 22 | **Disclosure of Confidential Data**<br>The disclosure of confidential data can occur if sensitive data can be viewed by unauthorised users | ALL | C |
| 23 | **Data Tampering**<br>Data tampering refers to the unauthorised modification of data | ALL | I |
| 24 | **Luring Attacks**<br>A luring attack occurs when an entity with few privileges is able to have an entity with more privileges perform an action on its behalf | WA3 WA4<br>WA5 WA6 | C<br>I |
| **Configuration Management** | | | |
| 25 | **Unauthorised Access to Administration Interfaces**<br>Malicious users able to access a configuration management function can potentially deface the Web site, access downstream systems and database | WA4<br>WA6 | C<br>I |

| | | | |
|---|---|---|---|
| 26 | **Unauthorised Access to Configuration Stores**<br>Because of the sensitive nature of the data maintained in configuration stores, you should ensure that the stores are adequately secured | WA4<br>WA6 | C<br>I |
| 27 | **Retrieval of Clear Text Configuration Data**<br>Sensitive data such as passwords and connection strings should be encrypt in that it helps prevent external attackers from obtaining sensitive configuration data | WA3 WA4<br>WA5 WA6 | C |
| 28 | **Lack of Individual Accountability**<br>Lack of auditing and logging of changes made to configuration information threatens the ability to identify when changes were made and who made those change | WA3 WA4<br>WA5 WA6 | Ac |
| 29 | **Over-Privileged Process and Service Accounts**<br>If application and service accounts are granted access to change configuration information on the system, they may be manipulated to do so by an attacker | WA4<br>WA6 | C<br>I |
| **Sensitive Data** | | | |
| 30 | **Access sensitive data in storage**<br>Sensitive data must be secured in storage to prevent malicious users from gaining access to and reading the data | ALL | C |
| 31 | **Network Eavesdropping**<br>An attacker uses network monitoring software to capture and potentially modify sensitive data | WA4<br>WA6 | C |
| 32 | **Data Tampering**<br>Data tampering refers to the unauthorised modification of data, often as it is passed over the network | ALL | I |
| **Session Management** | | | |
| 33 | **Session Hijacking** | WA4 | C |

| | | | |
|---|---|---|---|
| | A session hijacking attack occurs when an attacker uses network monitoring software to capture the authentication token (often a cookie) used to represent a user's session with an application | WA6 | |
| 34 | **Session Replay**<br>Session replay occurs when a user's session token is intercepted and submitted by an attacker to bypass the authentication mechanism | WA4<br>WA6 | C<br>I |
| 35 | **Man in the Middle**<br>A man in the middle attack occurs when the attacker intercepts messages sent between you and your intended recipient | ALL | C<br>I |
| **Cryptography** | | | |
| 36 | **Poor Key Generation or Key Management**<br>Attackers can decrypt encrypted data if they have access to the encryption key or can derive the encryption key | WA3 WA4<br>WA5 WA6 | C |
| 37 | **Weak or Custom Encryption**<br>Weak encryption algorithm provide no security if the encryption is cracked or is vulnerable to brute force cracking. Custom algorithms are particularly vulnerable if they have not been tested | WA3 WA4<br>WA5 WA6 | C |
| 38 | **Checksum Spoofing**<br>Some Hash algorithm can be interpreted and changed | WA3 WA4<br>WA5 WA6 | <br>C |
| **Parameter Manipulation** | | | |
| 39 | **Query String Manipulation**<br>The application is vulnerable to attack if the query string values represent sensitive data such as monetary amounts | WA3 WA4<br>WA5 WA6 | C |

| | | | |
|---|---|---|---|
| 40 | **Form Field Manipulation**<br>Form fields of any type can be easily modified and client-side validation routines bypassed | WA4<br>WA6 | C |
| 41 | **Cookie Manipulation**<br>Cookie manipulation is the attack that refers to the modification of a cookie, usually to gain unauthorised access to a Web site | WA4<br>WA6 | I |
| 42 | **HTTP Header Manipulation**<br>An attacker may have to write his own program to perform the HTTP request, or he may use one of several freely available proxies that allow easy modification of any data sent from the browser | WA3 WA4<br>WA5 WA6 | I |
| **Exception Management** | | | |
| 43 | **Attacker Reveals Implementation Details**<br>Internal implementation details such as exception details should not being reviewed by an attack which can greatly help them exploit potential vulnerabilities and plan further attack | WA3 WA4<br>WA5 WA6 | C |
| 44 | **Denial of Service**<br>Attackers will probe a web application, usually by passing deliberately malformed input | WA3 WA4<br>WA5 WA6 | A |
| **Auditing and Logging** | | | |
| 45 | **User Denies Performing an Operation**<br>The issue of repudiation is concerned with a user denying that he or she performed an action or initiated a transaction | ALL | Ac |
| 46 | **Attacker Exploits an Application Without Trace**<br>System and application-level auditing is required to ensure that suspicious activity does not go undetected | ALL | Ac |
| 47 | **Attacker Covers His or Her Tracks** | ALL | Ac |

| | | | |
|---|---|---|---|
| | Your log files must be well-protected to ensure that attackers are not able to cover their tracks | | |

### 5.2.2.3 Threat Elicitation Algorithm

In this section, the process of filtering threats from common threat list according to its web application type and security requirements is described in the following algorithm. Starting from the web application classification, each threat in common threat list is sieved by the rule, as a result, a threat list applying for the given web application can be obtained.

Just like described in [115], "a threat is any potential occurrence, malicious or otherwise, that could harm an asset". In other words, a threat is any bad thing that can happen to your assets. It is meaningless to discuss threats without connection to their assets. Hence, it is necessary to associate the threats to their comprised assets so that web application developer can design proper security mechanism to protect the assets.

**Threat Elicitation Algorithm**

| | |
|---|---|
| Input | Common ThreatList *CTL*<br>*WA*, is the web application type |
| Output | ThreatList *TL* |
| Initialisation | *TL*=∅ |
| Phase 1 | Procedure ThreatElicitation(*CTL*, *WA*) return *TL*<br>1. classify(*wa*);<br>{*Classify the given web application into one of the proposed web application type according to three attributes, use WAi to represent*}<br>2. rate();<br>{* Web application is rated "Low", "Medium", or "High" on the |

| | |
|---|---|
| | metrics of Confidentiality, Integrity, Availability, and Accountability use {CIAA requirements} to represent ***}**<br>3. for all threats $T_i$ in common threat list *CTL* do<br>4. if $T_i$. *WA Category == All* then<br>5. $T_i \rightarrow \{TL\}$ {*Insert $T_i$ to TL*}<br>6. end if<br>7. if $WA_i$. *WA Category* $\in T_i$. *WA type* then<br>8. $T_i \rightarrow \{TL\}$ {*Insert $T_i$ to TL*}<br>9. end if<br>10. end for<br>11. for *all* $TL_i$ in *TL* do<br>12. if TLi. CIAA risk does not match the {CIAA requirements} then<br>13. $TL_i \leftarrow \{TL\}$ {*Remove $TL_i$ from TL*}<br>14. end if<br>15. end for<br>16. connect ()<br>{* Establish the connection of each filtered threat to its target assets *} |
| Notes | *C*: Confidentiality *I*: Integrity<br>*A*: Availability *Ac:* Accountability |

List 5-1 Threat Elicitation Algorithm

### 5.2.2.4 Threat Risk Quantification

A threat list can be obtained by using the proposed threat elicitation method. Threats should be quantified in order to perform a comprehensive risk assessment for the target

system. In this case, DREAD [139] is used to rate the security risk for each threat. DREAD is part of a system for classifying computer security threats used at Microsoft. DREAD is a scheme used at Microsoft for classifying security threats. DREAD is an acronym for a set of principles that can used to estimate the overall risk for a given application. DREAD stands for:

- Damage Potential: what are the amount of potential damage caused by a successful attack
- Reproducibility: how easily the attack can be performed and repeated
- Exploitability: what are the skill level and resources required to perform the attack successfully
- Affected Users: how many users are affected if the attack is successfully executed
- Discoverability: how quickly and easily an occurrence of an attack can be identified

DREAD use numeric scheme to represent the serious level of each category. The calculation always generates a number between 0 and 10, the greater the number, the more serious the risk. Here is a use case [139] of how to quantify the DREAD categories in this study.

Table 5-8 DREAD Use Cases [139]

| Category | Use Cases | Value |
|---|---|---|
| Damage potential | Leaking trivial information | 0 |
| | Individual user data is compromised or affected | 5 |
| | Complete system or data destruction | 10 |
| Reproducibility | Very difficult to reproduce | 0 |
| | One or two steps required | 5 |
| | Just a web browser, without authentication | 10 |
| Exploitability | Very skilled | 0 |

| | | |
|---|---|---|
| | Malware or attack tool available | 5 |
| | Novice programmer | 10 |
| Affected users | None | 0 |
| | Some users | 5 |
| | All users | 10 |
| Discoverability | Unlikely | 0 |
| | Accessible only to few users | 5 |
| | Published | 10 |

### 5.2.3 Vulnerability Analysis

Once the potential threats are identified, a vulnerability analysis is performed to show what weaknesses exist in the system that may be exploited by the threat. The vulnerability analysis considers how to identify vulnerabilities and their potential impact of loss from a successful attack.

Existing vulnerability identification tools for application can be used to perform vulnerability identification and calculation of the severity scale produced. Vulnerability analysis aims at discovering the vulnerabilities within the application assets rather than data asset.

In this thesis, a vulnerability scanner for web application named N-Stalker [132] is chosen as the sample toolkit. N-Stalker provides a free edition to enhance the overall security level of web applications, using the most complete web attack signature database “N-Strealth Web Attack Signature Database”.

Common Vulnerability Scoring System (CVSS) [116] is one of the frequently used vulnerability scoring systems providing free and open standard quantification approach for scoring the severity of vulnerability and facilitates to determine the priority and urgency of response. The vulnerabilities identified by the N-Stalker will then be evaluated using CVSS. A total score of the vulnerabilities in the application can be produced. The scores are generated from a serious of metrics based on expert measurement involving base metric, temporal metric and environmental metric. For

each of metric, the score ranges from 0 to 10 which means that vulnerabilities with a base score in the range 7.0-10.0 are critical, those in the range 4.0-6.9 as major, and 0-3.9 as minor [116].

### 5.2.4 An Example

In order to illustrate the usefulness of web application classification, this section will examine a case study used in [152]. The case study is described as: “Widgets Incorporated is a medium-sized consumer goods company. They have determined the need to create I-Tracker: a custom-built inventory tracking application to facilitate growing customer demand. The most common use case will be for sales staff to enter data from a sales order which will automatically update the inventory levels and alert the logistics staff to prepare the order for shipment. When the inventory level for a particular widget drops below a certain threshold the manufacturing division will be notified. The main types of data used in the application include inventory levels, customer IDs, sales orders numbers, descriptions of orders, and product IDs. I-Tracker will be used by 30 internal users spread across the manufacturing, sales, and logistics departments, and that number is anticipated to grow to as much as 100 in the next few years. The business has indicated that the application may need to interface with a partner Widget Accessory supplier in the future. Widgets Incorporated currently receive 50-60 orders per day and anticipates that number grow to around 150.[152]”

According to the description of I-Tracker, data flow diagram can be drawn and shown in Figure 5-7.

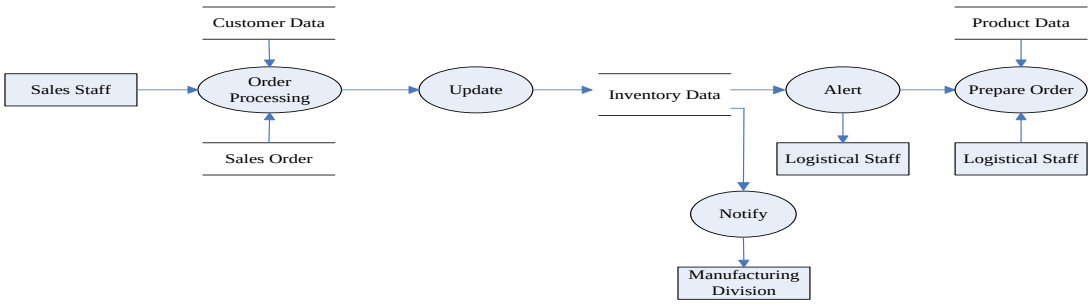



Figure 5-7 Data Flow Diagram of the I-Tracker

**Phase1. Architecture and Environment Analysis**

From the case description, the example web application is used within internal environment facing a number of known users. Thus, it belongs to the first type of web application categories.

**Phase2. Asset analysis**

Based on the description of section 5.2.1, assets can be identified as data store, data flow and processes. Process is the dynamic execution of application, while data flow is the dynamic representation of a "flow" of data in the system.

Table 5-9 Asset Criticality Analysis

| ID | Asset Name | Type | Sensitive | Environment | Criticality | Rating |
|---|---|---|---|---|---|---|
| 1 | Customer data | Internal Data | Yes | Internal use | Medium | 5 |
| 2 | Sales order | Sensitive Data | Yes | Internal use | Medium | 5 |
| 3 | Inventory data | Public Data | No | Internal use | Very Low | 1 |
| 4 | Product data | Public Data | No | Internal use | Very Low | 1 |
| 5 | Order processing | Sensitive process | Yes | Internal use | Medium | 5 |
| 6 | Order updating | Sensitive process | Yes | Internal use | Medium | 5 |
| 7 | Logistics staff alerting | Non-sensitive process | No | Internal use | Very Low | 1 |
| 8 | Inventory level notifying | Non-sensitive process | No | Internal use | Very Low | 1 |
| 9 | Preparing order | Non-sensitive process | No | Internal use | Very Low | 1 |

**Phase3. Threat analysis**

**Step1: Threat identification**

**1. Security requirement rating.** It is necessary to rate the security requirements CIAA of target application when using the proposed EDTEM. Using internal guidelines based on documents such as [30], the following application classification may be produced:

- Confidentiality: Low

  All data in the application is readily available to anyone in the company. Sensitive financial data and client private information are not handled by this application.

- Integrity: High

  Poor inventory and shipping tracking may result in significant financial loss to the company and may result in customer dissatisfaction / loss of customers.

- Availability: Medium

  A major disruption of the application will cause a backlog in shipping and have some financial consequences to the organisation. Minor disruptions, however, can be tolerated as customers expect a 4-6 week delay in receiving their goods.

- Accountability: Low

  Data processed in the system are not sensitive or personal. For the consideration of budget and time, system accountability is not included in the system design.

**2. Filtering.** According to the EDTED described in the previous section, the most likely threats are filtered and listed in Table 5-10.

Table 5-10 Threat List after Filtering

| ID | Threat Name | CIA |
|---|---|---|
| 6 | Viruses, Trojan horses, and Worms | CIA |
| 9 | Denial of Service | A |
| 11 | Unauthorised Access | CI |
| 20 | Credential Theft | CI |
| 21 | Elevation of Privilege | CI |
| 23 | Data Tampering | I |
| 35 | Man in the Middle | CI |

It can be inferred from the table above that internal attackers are the major factors to

perform the attacks.

**3. Further Filtering.** The threat list can be further screened out according to the security requirements of CIAA aspects. In terms of the EDTEM algorithm, threats with Confidentiality (C) and Accountability (Ac) requirements can be rule out in that the given application has low level requirements, while threats with Integrity (I) and Availability (A) are remained.

**Step2: Threat quantification**

DREAD is used to quantify the security level for each threat identified from our EDTEM according to the rating value in Table 5-9.

Table 5-11 Threat Risk Quantification

| ID | Threat Name | D | R | E | A | D | Total |
|---|---|---|---|---|---|---|---|
| 6 | Viruses, Trojan horses, and Worms | 10 | 10 | 10 | 10 | 10 | 10 |
| 9 | Denial of Service | 10 | 5 | 5 | 10 | 0 | 6 |
| 11 | Unauthorised Access | 5 | 5 | 5 | 5 | 0 | 4 |
| 20 | Credential Theft | 10 | 0 | 5 | 5 | 5 | 5 |
| 21 | Elevation of Privilege | 5 | 5 | 0 | 5 | 0 | 3 |
| 23 | Data Tampering | 10 | 5 | 5 | 5 | 5 | 6 |
| 35 | Man in the Middle | 5 | 5 | 5 | 5 | 5 | 5 |

**Phase4: Vulnerability analysis**

Using the vulnerability scanning tools, vulnerabilities can be identified and quantified with an overall evaluation number between 1 and 10 by using CVSS.

**Phase5: Potential risk**

From previous steps, the average score for asset and threat is 2.78 and 5.5 respectively. Suppose the vulnerability score of target application is 3 for simplicity.

According to Formula 5, security vector of the target application SV= $\sqrt{\left(A^2+T^2+V^2\right)/3} = \sqrt{\left(2.78^2+5.5^2+3^2\right)/3} \approx 3.84$. For a given qualitative metric

shown in Table 5-12, it can be concluded that the security risk of target web application is low.

Table 5-12 Risk Rating Scale

| Value | Treat Severity | Rank |
|---|---|---|
| 0.0-1.99 | Very low | 1 |
| 2.0-3.99 | Low | 2 |
| 4.0-5.99 | Medium | 3 |
| 6.0-7.99 | High | 4 |
| 8.0-10.0 | Very high | 5 |

## 5.3 Security Evaluation

Each security attribute requires one or more trusted mechanisms that are implemented in software components [170]. Security mechanisms mitigate the threats posed by exploiting vulnerabilities. As shown in Figure 5-8, a security evaluation method is proposed to assess whether the built-in security mechanisms in the legacy system satisfy the security requirements when facing elicited threats, which results in a decision whether the legacy system needs to be evolved to satisfy the user's security needs.

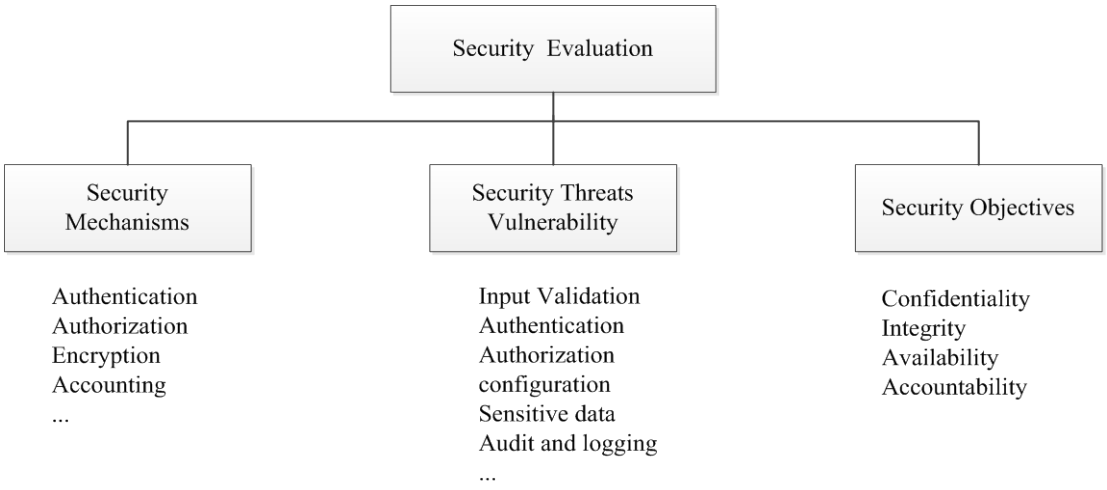



Figure 5-8 Security Evaluation Method

Let's consider the security mechanisms provided in I-Tracker case study in Section 5.2.4. Suppose the security mechanisms provided in I-Tracker are similar to those of the case study described in Section 4.6.4, Table 5-13 lists the result of security evaluation.

Table 5-13 Security Evaluation for I-Tracker

| Security Objectives | Security Threats | | | | Security Mechanisms | |
|---|---|---|---|---|---|---|
| CIAA Requirements | Threat ID | Threat Name | CIAA | Risk level (0--10) | Detected Security Mechanisms | Security level |
| Confidentiality(C): Low Integrity(I): High Availability(A): Medium Accountability(Ac): Low | T6 | Viruses, Trojan horses, and Worms | CIA | Very High (10) | | |
| | T9 | Denial of Service | A | High (6) | | |
| | T11 | Unauthorised Access | CI | Medium (4) | Username and password | Low |
| | T20 | Credential Theft | CI | Medium (5) | Well-proven encryption algorithms with enough encryption key | High |
| | T21 | Elevation of Privilege | CI | Low (3) | Group grants | Low |
| | T23 | Data Tampering | I | Medium (6) | Well-proven encryption algorithms | High |
| | T35 | Man in the Middle | CI | Medium (5) | Well-proven encryption algorithms | High |

## 5.4 Security Requirements Elicitation

Security requirements represent the types and levels when attempts to protect the assets to meet security policy [98]. A complete and consistent group of security requirements can be produced by using an elicitation method. Specially, security requirements are identified by risk analysis—"the systematic use of information to identify sources and to estimate the risk" [80]. As discussed in the previous section of this chapter, threat identification is one of the key parts during performing the risk analysis. Shown as Figure 5-9, threat modelling is one of the effective methods to elicit security requirements which can figure out the threats to the application security at the early stage. Security requirement elicitation for web application is proposed in the previous section due to its representativeness and complexity.

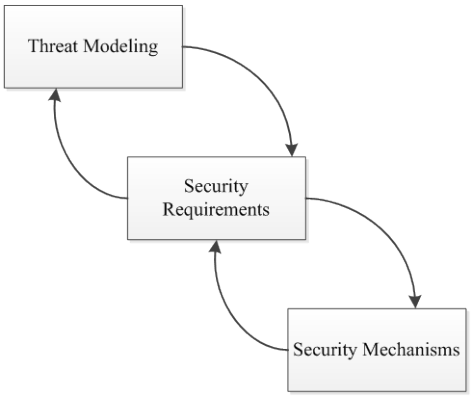



Figure 5-9 System Security Engineering [129]

Usually most functional requirements are specified as what must happen, while security requirements are stated in terms of what must not be allowed to happen. After the risk analysis in the previous section, assets can be enumerated with criticality level, threats threatening the assets can be elicited with severity risk level, security features that would be violated by potential threats on assets can be analysed, and priority level representing the developing order of the security requirements can be computed.

In this thesis, security requirements are elicited by risk analysis and represented as a list with columns of asset, threat, security features and priority, which means, for a given asset, the threats threatening to it and the severity level the threats may cause to the asset. It can be used as the priority order of security requirements when system

designers develop security aspects to satisfy them. Table 5-14 shows an example of security requirement format in this thesis.

Table 5-14 Example of Security Requirements

| SR No. | Asset | Threat | CIAA | Priority |
|---|---|---|---|---|
| SR1 | User bank account | Sniffing | Confidential | High |
| SR2 | User account | Cross-site Scripting | Confidential | High |
| SR3 | Place order | User Denies Performing an Operation | Accountability | Low |
| SR4 | Display product | Denial of Service | Availability | Medium |
| SR5 | Product Catalogue | Data tampering | Integrity | Medium |

## 5.5 Summary

Identification and evaluation of threat and assets are of great significance during security evolution. Identifying threats tells the security engineers how many and what kind of threats threatening the system, while evaluation shows to what extent the threats may do harm to the assets. By identifying and evaluating key threats and assets of the target system, it provides the detailed security requirements specifying which asset facing what kind of threats and resulting in what degree of the security risk to the system.

In order to identify and assess the risks the legacy system faced and the security mechanisms built in in a legacy system, this chapter proposed an assessment approach to evaluate the security risks level of the legacy system. The contents covered in this chapter are concluded as follows:

- For security requirements elicitation, a risk assessment framework based on security vector <threat, asset, vulnerability> is proposed as the means to evaluate the risk level a legacy system faced.
- Method for asset identification and asset criticality quantification is described

- A method to web application classification is proposed as the foundation to modelling threat for web applications. The environment where the web applications host is taken into account.
- Threat modelling is the most important part during the security requirement elicitation. A threat elicitation method is presented based on the proposed environment web application classification, which is a light weight and easy to use especially for security novice users.
- Security evaluation for legacy system is the prerequisite for security-driven software evolution. With the detection of security implementation in the legacy system and the security objective from the users, the system designers can make the decision whether or not the current security implementation satisfies the security requirement and whether or not the security evolution is required.

# Chapter 6

# Security Enhancement in Evolution

**Objectives**

- To illustrate the framework for security enhancement in evolution
- To illustrate the method to organise the security patterns
- To define mechanisms to automate search security patterns to fulfil the security requirements
- To illustrate the integration of security patterns with the system models

## 6.1 Overview

Experience shows that it is difficult to design a system that can fulfil the specific security requirements by simply integrating security mechanism and be error free at the same time, even for a small system. Security expertise tends to be valuable in such circumstances. However, such security expert knowledge is not always available for ordinary software developers. What's more, with the systems are getting larger and complicated, which makes the situation getting worse.

Inspired by design patterns, security patterns incorporate security expertise to solve the security problems occurred in the specific contexts. For security novice, security patterns represent security best practices which are a convenient way to design secure and reusable software systems [150]. They document basic mechanisms, process or approaches which provide ways to safeguard CIA features of data [151].

In this thesis, the security problems of legacy system are addressed through the use of security patterns. Patterns are well-known solutions to common problems in the given

contexts. Pattern writers have introduced many collections of security patterns recently (see the Chapter 2 “related work on Security Pattern”). However, there are some features missing which impedes the benefit of taking advantage of security patterns. One of the most fundamental features is how to find the appropriate from the existing security pattern to solve the given security problems.

It is not possible to get right and meaningful answers automatically because of no syntax defined in security patterns [150]. Therefore, a framework for semantic description and management of security patterns by the light of defining proper security ontology is developed in this chapter. The fundamental idea is to ease the searching of “right” security patterns with the help of ontology technique. Security patterns can be descibed based on the proposed security core ontology which describes security patterns semantically and precisely. Therefore, sophisticated retrieval and search of security patterns are enabled.

## 6.2 Framework of the Security Enhanced Approach

In this section, an overview of the proposed security enhancement framework is shown in Figure 6-1. All activities in the framework form the following sections of this chapter. The input of the security enhanced approach includes system design model extracted from Chapter 4, security requirments conceptual model derived from Chapter 5 and existing security patterns. The outputs of the proposed approach are security enhanced design models which can be further transformed into code by using MDA automation tools.

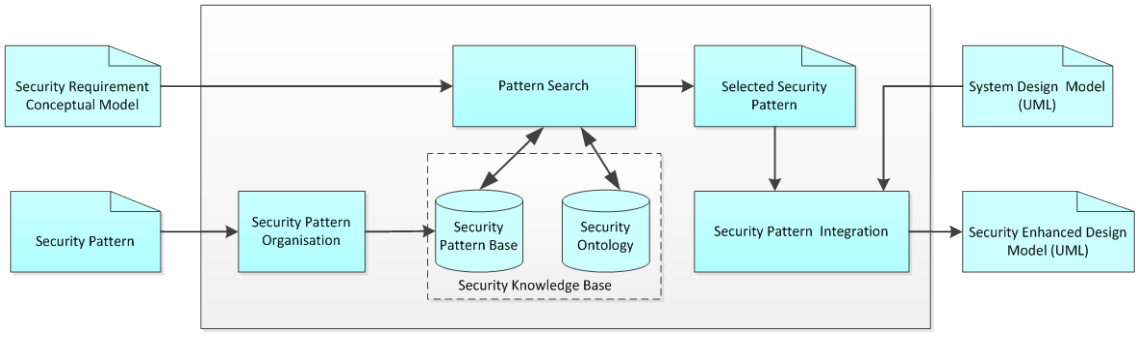



Figure 6-1 Operational Framework for Chapter 6

In the proposed approach, security patterns are represented with specific profiles and

solutions for various environments. The descriptions of security pattern include abstraction level, type of solution, applicability, context conditions and security properties provided by the pattern. A series semantic properties is defined to each pattern, such as “security attribute: Confidentiality”, “Deployed in design phase” …and so on. The incorporation of precise and rich semantic descriptions of the security patterns enables the use of automated reasoning mechanisms capable of searching proper patterns to fulfil the given security requirement.

In order to achieve this goal, a security ontology is developed and stored as one part of security knowledge base together with security pattern organised as the other one. Figure 6-2 shows the structure of the proposed security knowledge base. A pattern search engine is designed to find the right security pattern with the proposed security ontology inference to satisfy the corresponding security requirements.

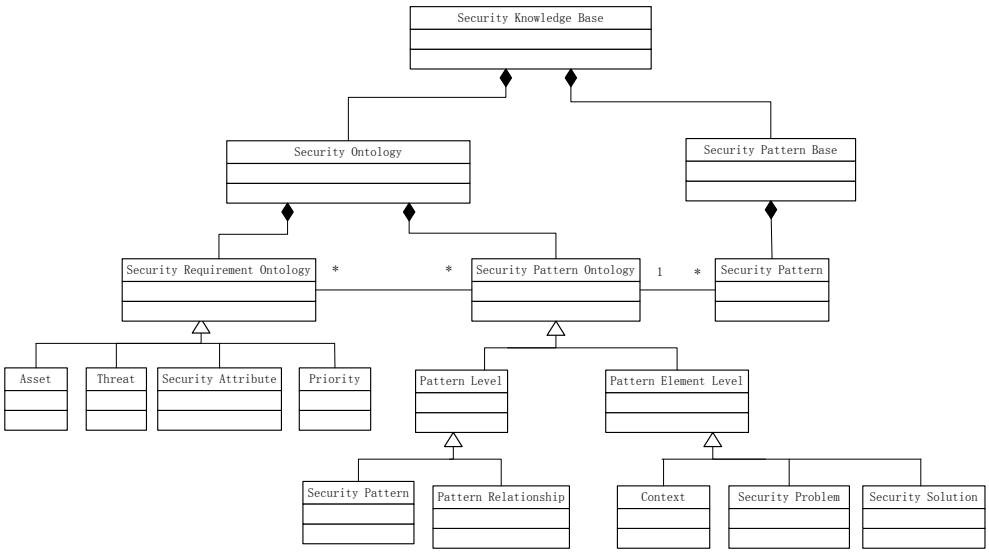



Figure 6-2 Class Diagram for the Security Knowledge Base Meta-model

Basically, Figure 6-2 shows that the structure of the security knowledge base is similar to a tree structure for storing security related information that helps to reveal and organise the security relevant features, and for relating these properties to fundamental security requirements. It consists of two sub repositories, security ontology base and security pattern base. Security ontology base is used to store the established ontology including concepts and relationships while security pattern base is the repository to store

and organise the common security patterns for further processing. Considering the reusability of the stored security relevant information, the information is expressed in a generalised way and focuses on abstract level.

It is impossible to develop a completely secure system because of the cost, time, and resources needed for the development and the emerging new kinds of attacks, even if it could be done, the usability and efficiency of the developed system may be decreased. Thus, developing secure systems is about trade-offs and it is quite a challenge to find a balance point. Prioritising of each elicited security requirement and incorporating user's security objectives play a key role when facing such a dilemma.

The criticality of each asset has to be decided, which implies a criterion for the security threshold of an asset is decided according to not only the value impact but also the risk for the asset, including likelihood and impact. Therefore, analysing the threat and vulnerability of a system in order to evaluate the risk is required. Specially, analysis of the threats threatening to the system is used as s a means of identifying why the assets need protection. In addition, the vulnerability of a system is analysed in order to understand what weakness exist in the system that can be exploited by the threats. This is the process of security requirement elicitation. The outcome of this process will be a list of security requirements with priorities representing their criticalities to the system. The process is established during the security requirement analysis by using the proposed EDTEM in Chapter 5.

## 6.3 Security Pattern

Security patterns incorporate proven security expertise solution to the recurring security problems. Usually, the security requirements are addressed by abstracting the security problems arising in a specific context and providing a well proven solution to them [158]. The ability to mitigate and stop security threats of security patterns can be found in [69, 70] which means that security patterns incorporated into the system could contribute to the system's security level [49].

It should be noted that security patterns can be designed and developed by security experts for different kinds of problem solving and be applied in different contexts. For

example, they can be abstract higher level architectural patterns that specify how to resolve a security problem architecturally, or they can be even more abstract patterns that depict the process to secure software development, or they can be defensive design level patterns describing how the detailed security artefacts can be created [158].

### 6.3.1 Security Pattern Format

The documentation of security patterns were originally built by Yoder et al. [184] in 1997. Seven architectural security patterns are presented and structured using the formats in POSA [21] or GoF [53] which are generic scheme for describing design patterns in architecture level.

The format is composed of several elements shown as follows [26]:

- *Intent*: description of goal and issues the pattern addresses;
- *Context*: description of situations or environment in which the pattern is used;
- *Problem*: description of the problem that this pattern solves;
- *Description*: description of the scenarios that illustrate the design problem;
- *Solution*: description of the solution to the problem;
- *Consequences*: description of the trade-offs and results when this pattern is used;
- *Forces*: description of constraints that should be considered when the pattern is applied.
- *Known uses*: description of the patterns use found in real systems;
- *Related patterns*: description of the related patterns that use this pattern as a reference.

In the view of pattern format, pattern authors can describe all sections which they consider of importance. Therefore, to detect proper security, just *Problem* and *Context* elements will be used as security pattern problem section patterns from a security point of view.

The structure of security patterns adopted in this thesis is based on the traditional design patterns. They have an expressive name, an application context, to be solved problem

and a solution to that problem.

Therefore, security pattern is represented as a 3-tuple <*Context, Problem, Solution*> where:

- *Context* defines the conditions and situation in which the pattern is applicable

  Time and location are usually regarded as important characteristics of context in the security domain. Time relates to when a security problem occurs and the location specifies at which level of system infrastructure a security problem occurs. In terms of software domain, typical example of the time within a context is software life cycle phases which are analysis, design, implementation, integration, and location where the operation occurs usually expressed as application, host and network [150].

- *Problem* defines the vulnerable aspect of an asset

  The problem field of a security pattern is important for software developers to determine whether a security pattern is appropriate for their situation. This field defines the security problems occur in the specific contexts and can be solved by the security pattern. A security problem occurs whenever a system is unprotected or is protected insufficiently against abuse or misuse. Generally speaking, security problem can be some kinds of threats which cause possible danger or damage when someone or something violates security policies.

- *Solution* defines the scheme that solves the security problem which occurs in the security context

  Security solution is a group of one or more countermeasures which have to be applied in order to mitigate the risk. For each threat there should exist at least one security countermeasure.

### 6.3.2 Security Pattern Relations

Experience shows that some security solutions may introduce new security problems, and in some cases only a part of the problem might be solved by using them. Under such circumstances, additional security patterns should be taken into account. That it is

why some security patterns should be applied along with other patterns in reality. For example, Single Access Point pattern requires Check Point as a prerequisite. Figure 6-3 shows how the access control patterns are related to each other [151]. Therefore, it is necessary to identify and explicitly consider their relationship to effectively select a set of patterns.

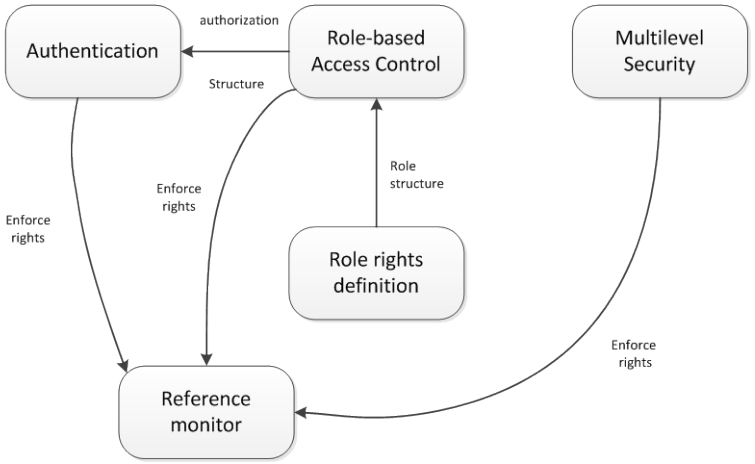



Figure 6-3 Access Control Patterns Relationship [151]

There can be various types of relations among security patterns. However, as described in [150], "all pattern relations can be expressed in terms of the primary relationships namely *refines*, *uses* and *conflicts"*. Two relations *refines* as *specialises* and *uses* as *requires* are taken into consideration in this thesis. The definition of the two relations in [150] is adopted in this thesis:

- Specialises relations. It usually used to make an abstract pattern more specific in terms of the context and problem domain. For example, Role Base Authentication pattern is the specialisation of Authenticator pattern with more specifications.
- Requires relations. It specifies that another security pattern is needed during the application of a security pattern. This may be caused by the following reasons. Firstly, a single pattern is not enough to address the security problem caused by a particular threat or attack and another pattern is needed as a complement. Secondly, the application of a pattern may lead to a new security problem. For

example, Single Access Point pattern requires Check Point as a prerequisite.

### 6.3.3 Security Pattern Organisation

A significant number of security patterns have been proposed since the first effort in 1977 by Yoder et al. [184]. A security pattern may address more than one security feature, for example, Authentication pattern can protect both confidentiality and integrity security features. At the same time, for a specific security property, there may be more than one security pattern can address it. It is a many-to-many situation. Additionally, security patterns may be organised by different parameters from abstract to more specific. Hence, it is difficult to find the "right" security patterns for solving a particular security problem without a proper classification scheme of security pattern [65]. A suitable classification scheme not only contributes to efficient information storage and retrieval, but also benefits both pattern navigators and pattern miners.

In this section, on the basis of several existing classification frameworks, an efficient classification framework for security patterns has been described to facilitate finding the proper security patterns according to the elicited security requirements in Chapter 5. As the security requirement is based on threat modelling and asset analysis, the properties of threat and asset will be considered as the factors for selecting security patterns. The proposed classification scheme is based on multiple aspects of the relevant information.

- Lifecycle Stage.

  While most of the security patterns take the form derived from design patterns, not all security patterns are dedicated to design phase. Therefore, classification on the lifecycle stages are meaning for organising security patterns ordered on the dichotomy of beginning and end, which are: Analysis, Architecture, Design, Implementation, and Deployment.

- Architectural Layer.

  Layer provides another useful dimension, since problems and their solutions in different layer of the architecture differ, yet all are important. Roughly, the architecture has been divided with an ordering from low to high level of abstraction. The following distinctions are used as the architecture layers, which

are: Data, Application, System, and Network.

- Application Context.

  Another classification factor considers the structure of the system which is called Application Context and partitions the patterns according to which part of the system they are trying to protect [65]. The security of a system is analysed from three levels: core security, perimeter security and exterior security. The core security deals with the security implementation within the system while the perimeter security focuses on security related issues at the system entry points, such as authorisation, authentication and security. The exterior security considers protecting data during transmission and securing communication protocols.

- Domain Specific

  Application domain can provide an important differentiator or filter to narrow the field of applicable knowledge [167]. Some security pattern solutions are specific to a particular domain or application type. This dimension is an exception in that it does not have a dichotomy or ordering—the space is freely defined. Pattern designers can create patterns for their own domain as a form of knowledge capture. After examining the existing security pattern, several example domains are provided in this thesis: Ubiquitous computing, Distributed computing, Web and J2EE, Embedded system, Operating system, Service oriented architecture, SCADA (Supervisory Control and Data Acquisition), and not limited to this coverage.

- Threat Type.

  The classification scheme based on threat modelling is more intuitive because it uses the security problems that the patterns solve. Security architects use threat modelling to identify and prioritise a system's security threat. This let them prioritise the mitigation effort. STRIDE [160] is one of the widely used models to classify threats according to different sources, and their relevance to security features is shown in Table 6-1. It is the English acronym of the following six threat types [160]:

  - Spoofing is someone or something masquerades to be legitimate and valid.

- Tampering is data interfered or modified during network communication.
- Repudiation is the situation that user denies performing a certain action which could be illegal and harmful.
- Information disclosure is when an unauthorised user gets access to confidential information, which he or she is not supposed to have access to.
- Denial of service is basically when a service is brought down intentionally or unintentionally resulting in the disruption of normal services for legitimate users.
- Elevation of privilege is when an unauthorised user gets higher privilege access from the one he or she was supposed to have, which might result in access to restricted information, or might apply dangerous tasks.

Table 6-1 Threat and Security Features [64]

| Threats Type | Security Features |
|---|---|
| Spoofing | Authentication |
| Tampering | Integrity |
| Repudiation | Non-repudiation |
| Information disclosure | Confidentiality |
| Denial of service | Availability |
| Elevation of privilege | Authorisation |

- Security Concerns

Software patterns are usually chosen by developers with a particular goal in mind. Developers tend to view security in terms of software requirements rather than taking the perspective of an attack. Therefore, it is necessary to employ security goals or concerns to classify the security patterns. This metric is more straight and easier understood to software designer to select proper security patterns in their security design. The security concerns used in this thesis are: Access control,

Authentication, Filtering, Confidentiality, Integrity, Availability, Accountability, and Non- repudiation.

Table 6-2 summarises the classification scheme for security patterns, based on which the security patterns repository with security relevant format <*Context, Problem, Solution*> is shown in Appendix A.

Table 6-2 Summary of the Proposed Multiple Aspects Classification Scheme

| Criteria | Classification | | |
|---|---|---|---|
| Lifecycle Stage | Analysis | Architecture | Design |
| | Implementation | Deployment | |
| Architectural Layer | Data | Application | System |
| | Network | | |
| Application Context | Core | Perimeter | Exterior |
| Domain Specific (Not limited) | Ubiquitous computing | Distributed computing | Web and J2EE |
| | Embedded system | Operating system | SOA |
| Threat Type (STRIDE) | Spoofing | Tampering | Repudiation |
| | Information disclosure | Denial of service | Elevation of privilege |
| Security Concerns | Access control | Authentication | Filtering |
| | Confidentiality | Integrity | Availability |
| | Accountability | Non-repudiation | |

## 6.4 Security Ontology

An ontology, in the field of knowledge representation, is most often defined as “a representation of a conceptualisation” [61]. A more detailed description of ontology is that “it is a formal representation of the entities and relationships which exist in some domains, it should also represent a shared conceptualisation in order to meet any useful

purpose" [35]. Ontologies are useful for representing and inter-relating many kinds of knowledge. In 2003, Marc Donner urged the necessity of having good security ontologies. He argued that too much security terminology is vaguely defined, thus it becomes difficult to communicate between colleagues and, worse, confusing to deal with the people we try to serve: "What the field needs is an ontology – a set of descriptions of the most important concepts and the relationships among them. A great ontology will help us report incidents more effectively, share data and information across organisations, and discuss issues among ourselves" [36].

The advantages of applying ontology technology into the information security domain are specified in [142] from three viewpoints: (1) ontologies can eliminate the ambiguity of items to a properties list and organise information in a systematic way at detailed level; (2) ontological technology can induce the modularity which can be used by other approaches, for example, to detect some new features by establishing relations among different measurements; and (3) an ontological approach has the ability to forecast security problems by providing inference mechanisms.

The approach proposed in this thesis can be summarised by the following. The security patterns for software engineering are created to document the knowledge of the experts in security field. These patterns are designed by using the ontology techniques that provide reusable and structured activities or solve security problems that can arise during the development of software systems. Moreover, due to the OWL representation, the security patterns are available in a machine readable format and it is expected to be utilised in the system automatically.

This chapter addresses the issue of fulfilling security requirements elicited in the previous chapter. The approach uses ontologies as a tool for managing different security requirements and associating them with corresponding security solutions provided by security patterns.

The main goal is to provide a security ontology based framework which unifies the proposed methods in security evolution for legacy system. The ontology "knows" which threats threaten which assets, and which security patterns could lower the probability of occurrence in which contexts. It is meaningful for the software developer to find the

appropriate security patterns by adopting an ontology based approach [38].

### 6.4.1 Existing Security Ontology

Even though several ontologies in information security domain have been proposed [36, 44, 76, 150], most of these approaches focus on technical aspects of security or aims at particular application domains. Therefore, they cannot be used directly since they don't specifying the general security model which can be used to describe security patterns.

The security ontology selected to be the basis of the proposed framework has been proposed by [44] which is developed based on the security relationship model described in the National Institute of Standards [81]. Figure 6-4 shows the high level concepts and corresponding relations of this ontology.

The four core concepts in [44] are: Asset, Threat, Vulnerability, and Control. Asset is something that has value to the Organisation and it requires certain level of security attribute. A threat is some kind of potential danger which threatens the asset with affecting the required security attributes. Each threat is specified by the threat origin such as human or natural, and by the threat source, deliberate or accidental. Vulnerability is the weakness in asset of the system that can be exploited by the threat, and a severity scale is used to represent the severity level caused if it executes successfully. A control is implemented to mitigate the vulnerability and protect the asset against corruption from threat. Control is derived from information security standard control and it is classified into several types [44].

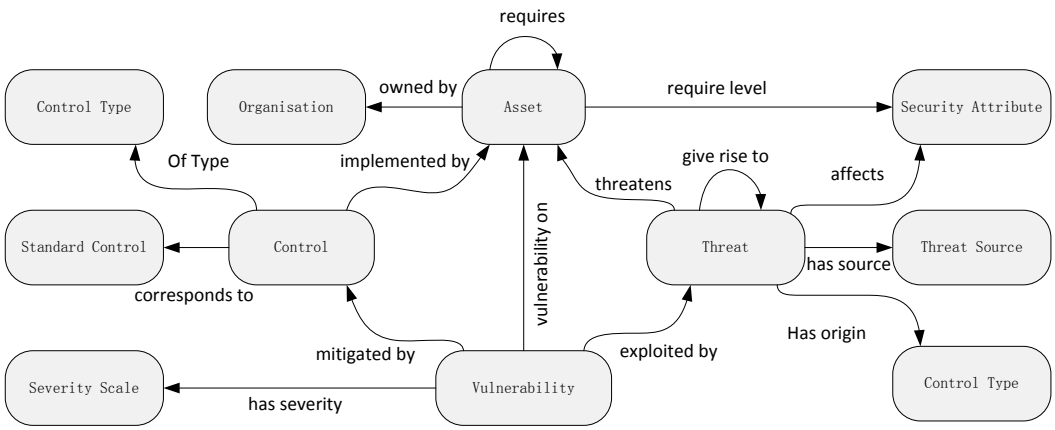



Figure 6-4 Security Ontology Top Level Concepts and Relationships [44]

### 6.4.2 Proposed Security Ontology

The proposed security ontology is designed to achieve the following goals:

- Describe risk relevant information especially security requirement information applicable to web application
- Design security pattern ontology at two abstraction levels
- Facilitate mapping security requirements to security pattern
- Provide the ability to annotate security related information to facilitate the security pattern selection
- Create reusable and easy to extend ontologies

The designed ontologies are supposed to be used by both the security pattern providers who design new security patterns and edit the corresponding ontology into the ontology base to express their security capabilities, and the security requirement requestors who have got security requirements to be fulfilled by security patterns. From the security requestor's point of view, security requirements can be stated in terms of 4-tuple <Asset, Threat, Security Attribute, Priority> which is elicited from the proposed risk assessment method in Chapter 5. From the security pattern provider's view of point, the security capabilities are expressed in terms of security patterns which are organised as 3-tuple <Context, Security Problem, Security Solution>.

The proposed ontology has been developed by using OWL, which is a language based on RDF for processing web information by the computer rather than being read by people. OWL is the current recommendation of W3C (World Wide Web Consortium) for processing the content of web information. OWL is a part of semantic web and has three sublanguages, OWL Lite, OWL DL (includes OWL Lite), and OWL Full (includes OWL DL). Based on Description Logics, OWL-DL has been used to design the proposed ontology for its expressivity is suitable for the requirement and allows for complete reasoning by DL reasoner, for example, Racer, FaCT++ or Pellet.

The tools used for developing and querying the security ontologies are Protégé and FaCT++. The Protégé Ontology Editor [140] provides the graphical interface for ontology designers to build OWL ontologies. However, the Protégé itself only provide

editing function and a reasoner (FaCT++ in this study) is required to check the consistency of the developed ontology.

#### 6.4.2.1 Security Ontology Development

Designing OWL ontology is not only defining a set of classes and properties, but also including a collection of restriction and axioms. This ensures that the correct result can be inferred from the proposed ontology.

There are several methods to develop ontology. The method used in this thesis is based on METHONTOLOGY [58] which is shown in Figure 6-5.

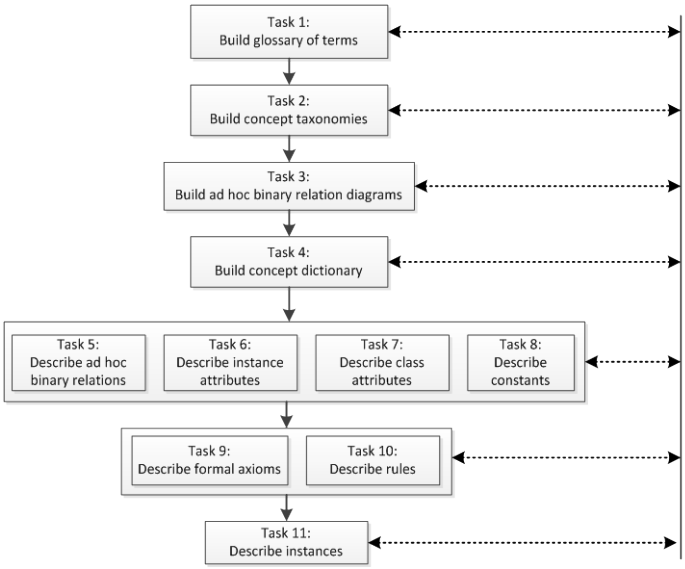



Figure 6-5 Tasks of the Conceptualisation Activity according to METHONTOLOGY [61]

The development of security ontology is carried out in the following phases:

- Define questions. A collection of questions within the domain is defined to indicate what kind of answers and information are expected by using the ontology. The questions are informal and loosely structure as any forms. Some important concepts can be identified during this process, which can be termed as the basis when building ontology classes.
- Build classes. Based on the previous phase, a lot of relevant concepts and terms have been identified and recorded. They can be classified and selected according to their relevancy to the domain to form the classes, or properties of the proposed ontology.

- Build Relationships. This process involves clarifying the relationships among the classes and defining the hierarchy. It is the process of adding axioms and restriction to the ontology. Axiom is a set of assertions specifying what is true in the domain. It is used to connect classes and properties with some logical information about them. Restriction is special kind of class description with that all individuals in that class will satisfy the restriction.
- Build ontology instance. This is the procedure to create instances of the classes which refers to inserting the individual information or providing examples of each of the classes.
- Validate ontology. The competence questions built in the first phase can be used to validate the correctness of the proposed ontology

The aforementioned phases have been repeated several times until the provided answers from the proposed ontology satisfy the competency question.

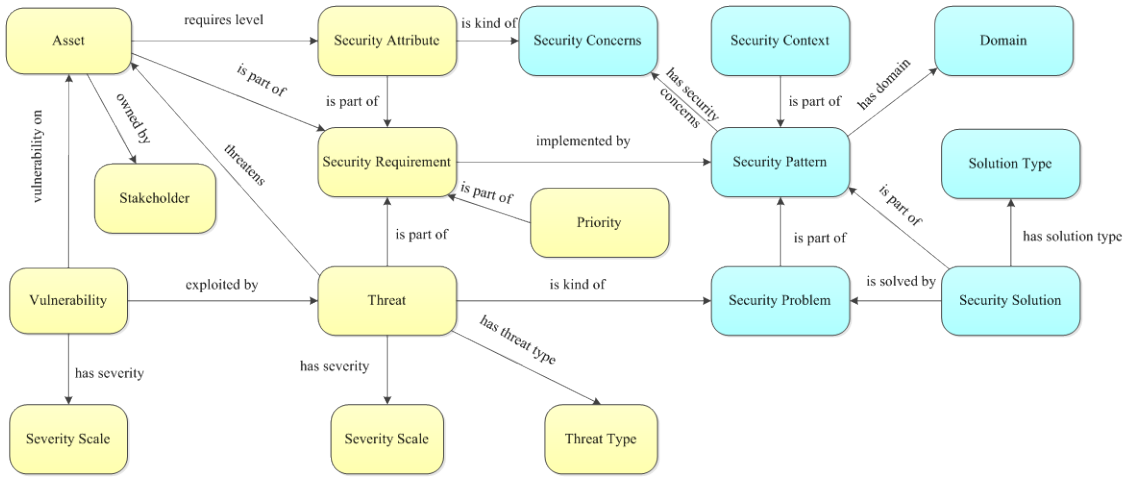



Figure 6-6 Proposed Security Ontology Top Level Concepts and Relations

Figure 6-6 shows the top-level concepts and relations of the proposed security ontology based on [44]. It is composed of two subontologies: security requirement subontology (sr) and security pattern subontology (sp). The security requirement subontology consists of the core concepts: Asset (sr:asset), Threat (sr:threat), Vulnerability (sr:vulnerability), Attribute (sr:attribute), Priority (sr:priority). The security pattern subontology is composed of the core concepts: Security Context (sp:context), Security

Problem (sp:problem), Secutity Solution (sp:solution). The concepts of sr:asset have been derived from [19], sr:vulnerability and sr:threat from [115], while security pattern subontology concepts are derived from [150].

### 6.4.2.2 Security Requirement Subontology

As described in Chapter 5, the security requirement is identified by risk analysis, which is one of the sources to elicit security requirement. Consequently, the requirement ontology (Figure 6-7) is developed with the concepts derived from the risk analysis using Protégé Editor shown in Figure 6-8. The meta-information associated with risk analysis (such as asset and threat) can be used to define axioms, constraints and rules that help to maintain the consistency of the proposed security requirement ontology.

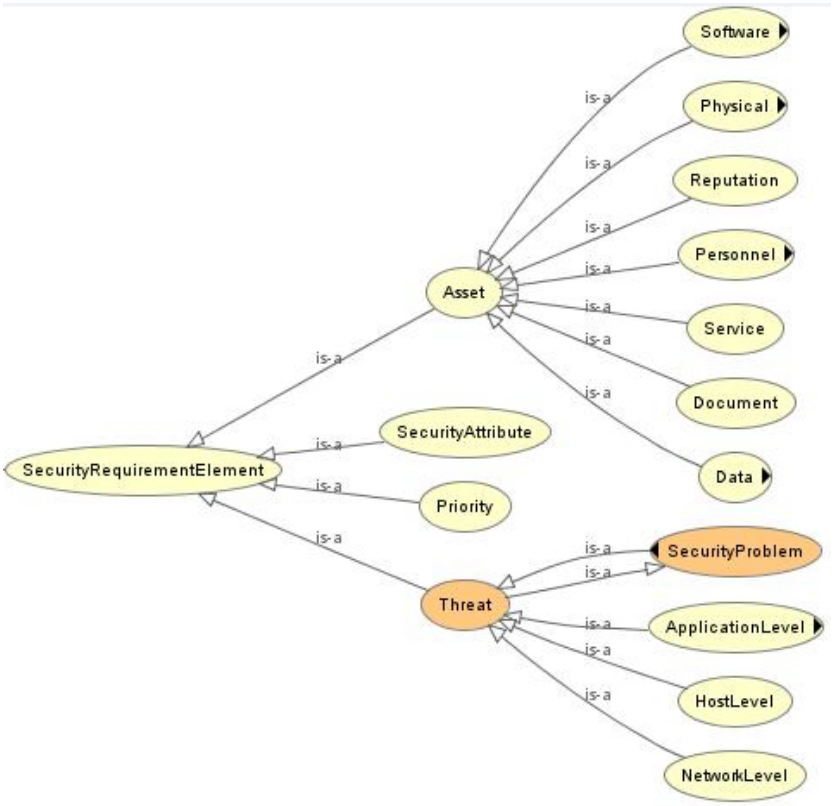



Figure 6-7 Top Level of Security Requirement Ontology

Every security requirement is a description of which asset is threatened by which kind of threat by violating which security objective and which severity extent of this requirement. The properties defined in security requirement ontology are described below:

- Each requirement is characterised by a unique identifier and has been defined as *Datatype property* in OWL.
- hasAsset: it represents the asset related to this requirement. It is defined as an *object property* with domain defined as class *SecurityRequirementElement* and range as class *asset*.
- hasThreat: it represents possible threats endanger the asset and then make the requirement unfulfilled. This property is represented by an *object property* and its range is the class *Threat* (Figure 6-9) defined in this ontology. There are constraints of which threat can be occurred to which asset according to the risk analysis in Chapter 5.
- hasSecurityAttribute: the feature that make an asset valuable. There exist four types of security properties using an object property: “Confidentiality”, “Integrity”, “Availability” and “Accountability”.
- hasPriority: the value can be computed from Formula 5 in Chapter 5 taking asset criticality, threat severity and vulnerability severity scale into account and shows the order of development. Datatype property {“high”, “Medium”, “Low”}.

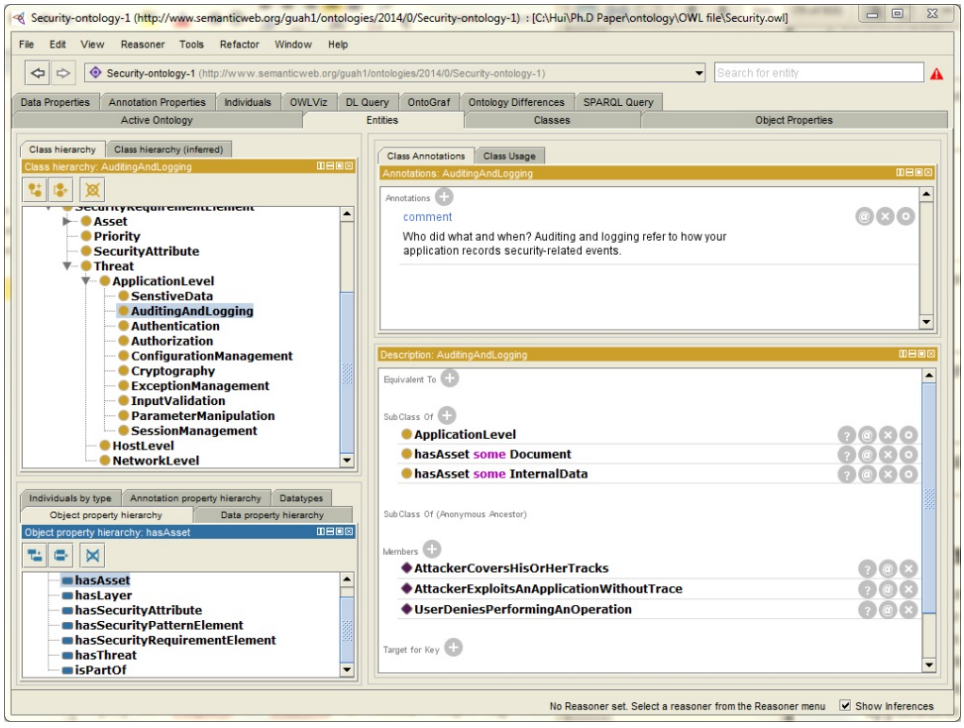



Figure 6-8 Taxonomy of the Elements of the Security Requirement Ontology in Protégé Editor

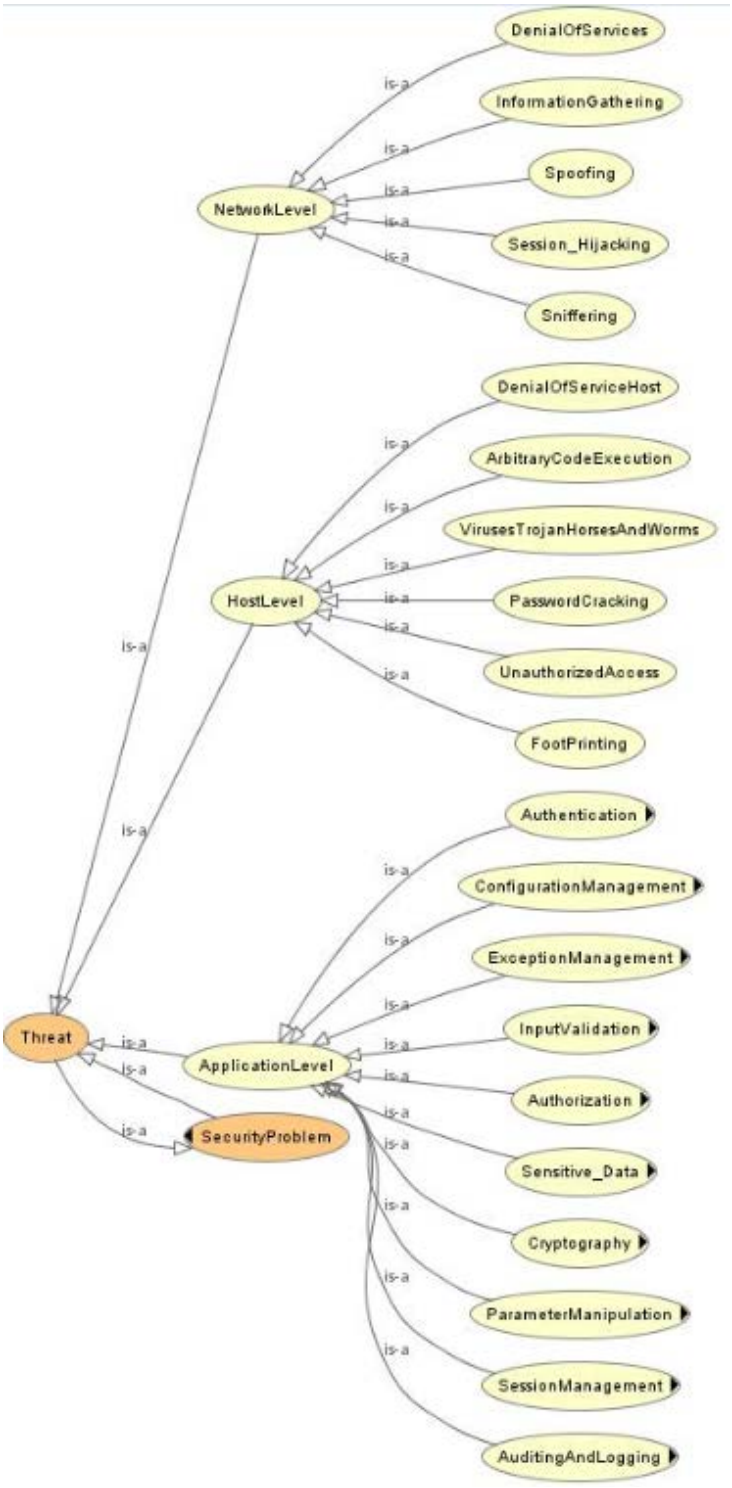



Figure 6-9 Top Level of Threat Ontology

List 6-1, List 6-2, and List 6-3 show the part of implementation of classes, object properties and instances in the proposed security requirement subontology in OWL code.

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Asset -->
    <owl:Class rdf:about="&Security;Asset">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </owl:Class>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Priority -->
    < owl:Class rdf:about="&Security;Priority">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </ owl:Class>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityAttribute -->
    < owl:Class rdf:about="&Security;SecurityAttribute">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </ owl:Class>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Threat -->
    < owl:Class rdf:about="&Security;Threat">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </ owl:Class>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ThreatType -->
    < owl:Class rdf:about="&Security;ThreatType">
        <rdfs:subClassOf rdf:resource="&Security;ClassificationKey"/>
    </owl:Class>
```

List 6-1 Top Level Classes Definition of Security Requirement Subontology

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#isThreatenedBy -->
    < owl:ObjectProperty rdf:about="&Security;isThreatenedBy">
        <rdfs:domain rdf:resource="&Security;Asset"/>
        <rdfs:range rdf:resource="&Security;SecurityProblem"/>
        <inverseOf rdf:resource="&Security;hasAsset"/>
    </owl:ObjectProperty>
```

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#residesOn -->
    < owl:ObjectProperty rdf:about="&Security;residesOn">
        <rdfs:range rdf:resource="&Security;Layer"/>
        <rdfs:domain rdf:resource="&Security;Threat"/>
        <inverseOf rdf:resource="&Security;containsThreat"/>
    </owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSecurityAttribute -->
    < owl:ObjectProperty rdf:about="&Security;hasSecurityAttribute">
        <rdfs:domain rdf:resource="&Security;Asset"/>
        <rdfs:range rdf:resource="&Security;SecurityAttribute"/>
    </owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasThreat -->
  <owl:ObjectProperty rdf:about="&Security;hasThreat">
      <rdfs:subPropertyOf rdf:resource="&Security;hasProblem"/>
  </owl:ObjectProperty>
```

List 6-2 Partial of Object Property Definition of Security Requirement Subontology

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DataTampering -->
    <NamedIndividual rdf:about="&Security;DataTampering">
        <rdf:type rdf:resource="&Security;Authorisation"/>
        <rdf:type rdf:resource="&Security;Sensitive_Data"/>
        <Security:residesOn rdf:resource="&Security;Application"/>
    </NamedIndividual>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DenialOfService -->
    <NamedIndividual rdf:about="&Security;DenialOfService">
        <rdf:type rdf:resource="&Security;ExceptionManagement"/>
        <rdf:type rdf:resource="&Security;HostLevel"/>
        <rdf:type rdf:resource="&Security;NetworkLevel"/>
```

```
            <rdf:type rdf:resource="&Security;ThreatType"/>
        </NamedIndividual>
    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Accountability -->
        <NamedIndividual rdf:about="&Security;Accountability">
            <rdf:type rdf:resource="&Security;SecurityConcern"/>
        </NamedIndividual>
  <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#NetworkEavesdropping -->
        <NamedIndividual rdf:about="&Security;NetworkEavesdropping">
            <rdf:type rdf:resource="&Security;Authentication"/>
            <rdf:type rdf:resource="&Security;Sensitive_Data"/>
        </NamedIndividual>
    …..
```

List 6-3 Partial of Instance Declarations of Security Requirement Subontology

#### 6.4.2.3 Security Pattern Subontology

As described in Section 6.3.1, the structure of the security pattern is a 3-tuple <Context, Problem, Solution> from the security point of view. Moreover, there are relationships among security patterns.

The security pattern subontology is based on [150], where the main properties are shown below:

- Security patterns are characterised by a unique identifier and a text description. Both have been defined in OWL as *Datatype properties*.
- hasContext: it represents the situation in which the security problem occurs and is defined as *object property*. The range of it is subclass SecurityContext. Two subproperties are hasLayer and hasLifeCycle whose ranges are Layer and LifeCycle respectively.
- hasProblem: it represents the security problem occur in such a security context and is defined as *object property*. The range of it is subclass SecurityProblem and an axiom is added as equivalent as subclass Threat in Security

Requirement subontology.

- hasSolution: it represents the security solution to the security problem that occurs in the given security context.
- hasThreatType: it represents the problem type classified according to threats with domain is SecurityPattern and range is ThreatType.
- hasSecurityConcerns: it represents the security features the security pattern holds.
- hasDomain: application domain the security pattern serves. It is defined as *object property* with domain is SecurityPattern and range is Domain.
- requires: it represents the Require relationship between security patterns. It is added as *object property* with the range is SecurityPattern.
- isSpecialisedBy: it represents the Specialise relationship between security patterns. It is added as *object property* with the range is SecurityPattern.

Figure 6-10 shows the illustration of security pattern subonotolgy and Figure 6-11 displays the screenshot of exemplifying the development of security pattern in Protégé Editor.

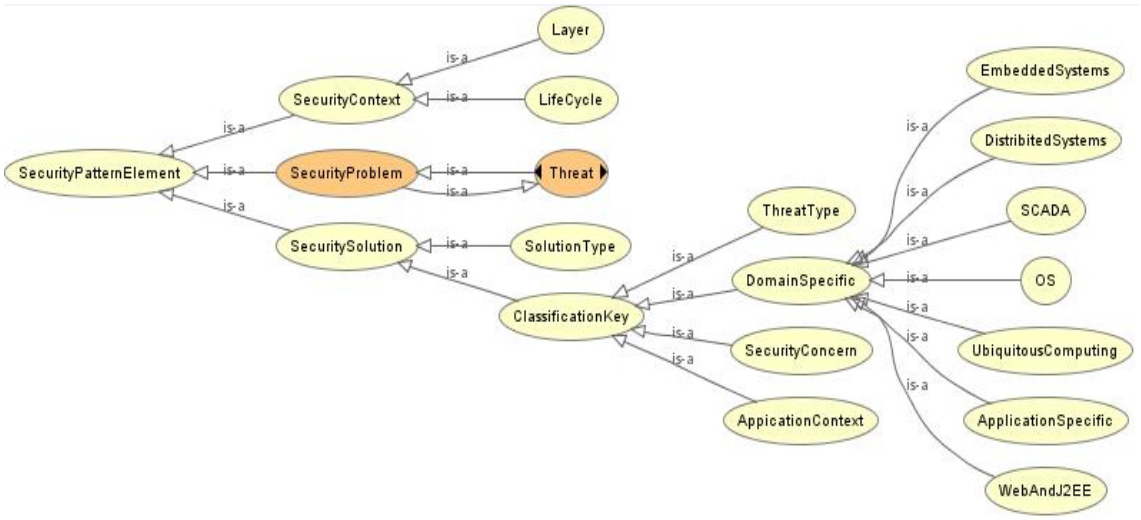



Figure 6-10 Top Level of Security Pattern Ontology

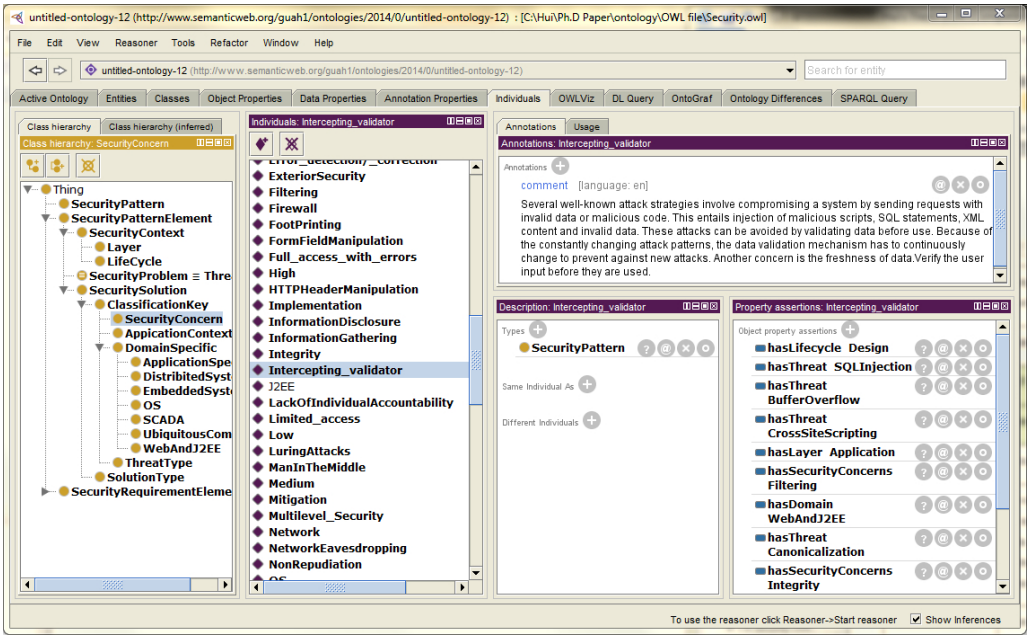



Figure 6-11 Screenshot of Intercepting Validator Pattern Implementation in Protégé Editor

List 6-4, List 6-5, and List 6-6 show the part of implementation of classes, object properties and instances in the proposed security pattern subontology in OWL code.

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityPattern -->
        <owl:Class rdf:about="&Security;SecurityPattern">
            <rdfs:subClassOf rdf:resource="&owl;Thing"/>
            <rdfs:comment xml:lang="en">A Security Pattern is a description of one particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution.</rdfs:comment>
        </owl:Class>
  <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityContext -->
        <owl:Class rdf:about="&Security;SecurityContext">
            <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
       </owl:Class>
 <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityProblem -->
        < owl:Class rdf:about="&Security;SecurityProblem">
            <equivalentClass rdf:resource="&Security;Threat"/>
```

```
            <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
        </owl:Class>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecuritySolution -->
    < owl:Class rdf:about="&Security;SecuritySolution">
        <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
    </ owl:Class>
```

List 6-4 Top Level Classes Definition of Security Requirement Subontology

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#containsThreat -->
        < owl:ObjectProperty rdf:about="&Security;containsThreat">
            <rdfs:domain rdf:resource="&Security;Layer"/>
            <rdfs:range rdf:resource="&Security;Threat"/>
        </ owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasApplicationContext -->
        < owl:ObjectProperty rdf:about="&Security;hasApplicationContext">
            <rdf:type rdf:resource="&owl;FunctionalProperty"/>
            <rdfs:range rdf:resource="&Security;AppicationContext"/>
            <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
        </ owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasDomain -->
        < owl:ObjectProperty rdf:about="&Security;hasDomain">
            <rdfs:range rdf:resource="&Security;DomainSpecific"/>
            <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
        </ owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasLayer -->
        < owl:ObjectProperty rdf:about="&Security;hasLayer"/>
        </ owl:ObjectProperty>
```

```
    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasLifecycle -->
        < owl:ObjectProperty rdf:about="&Security;hasLifecycle"/>
        </owl:ObjectProperty>
     <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasProblem -->
        < owl:ObjectProperty rdf:about="&Security;hasProblem"/>
        </owl:ObjectProperty>
  <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSecurityConcerns -->
        < owl:ObjectProperty rdf:about="&Security;hasSecurityConcerns"/>
        </owl:ObjectProperty>
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSolutionType -->
        <owl:ObjectProperty rdf:about="&Security;hasSolutionType">
            <rdf:type rdf:resource="&owl;FunctionalProperty"/>
            <rdfs:domain rdf:resource="&Security;SecuritySolution"/>
        </owl:ObjectProperty>
```

List 6-5 Partial of Object Property Definition of Security Requirement Subontology

```
        …….
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authentication -->
    <NamedIndividual rdf:about="&Security;Authentication">
        <rdf:type rdf:resource="&Security;SecurityConcern"/>
        <Security:hasLayer rdf:resource="&Security;Application"/>
        <Security:hasThreat rdf:resource="&Security;DataTampering"/>
        <Security:hasLifecycle rdf:resource="&Security;Design"/>
        <Security:hasSecurityAttribute rdf:resource="&Security;Integrity"/>
    </NamedIndividual>
        ……
  <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Intercepting_validator -->
      <NamedIndividual rdf:about="&Security;Intercepting_validator">
```

```
        <rdf:type rdf:resource="&Security;SecurityPattern"/>
        <rdfs:comment xml:lang="en">Several well-known attack strategies involve compromising a system by sending requests with invalid data or malicious code. This entails injection of malicious scripts, SQL statements, XML content and invalid data. These attacks can be avoided by validating data before use. Because of the constantly changing attack patterns, the data validation mechanism has to continuously change to prevent against new attacks. Another concern is the freshness of data.Verify the user input before they are used.</rdfs:comment>
        <Security:hasLayer rdf:resource="&Security;Application"/>
        <Security:hasThreat rdf:resource="&Security;BufferOverflow"/>
        <Security:hasThreat rdf:resource="&Security;Canonicalisation"/>
        <Security:hasThreat rdf:resource="&Security;CrossSiteScripting"/>
        <Security:hasLifecycle rdf:resource="&Security;Design"/>
        <Security:hasSecurityConcerns rdf:resource="&Security;Filtering"/>
        <Security:hasSecurityConcerns rdf:resource="&Security;Integrity"/>
        <Security:hasThreat rdf:resource="&Security;SQLInjection"/>
        <Security:hasDomain rdf:resource="&Security;WebAndJ2EE"/>
    </NamedIndividual>
    …..
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_pipe -->
    <NamedIndividual rdf:about="&Security;Secure_pipe">
        <rdf:type rdf:resource="&Security;SecurityPattern"/>
        <Security:hasThreat rdf:resource="&Security;BruteForceAttack"/>
        <Security:hasThreat rdf:resource="&Security;CookieReplay"/>
        <Security:hasThreat rdf:resource="&Security;CredentialTheft"/>
        <Security:hasThreat rdf:resource="&Security;DictionaryAttack"/>
        <Security:hasThreat rdf:resource="&Security;NetworkEavesdropping"/>
    </NamedIndividual>
      .....
```

List 6-6 Partial of Instance Declarations of Security Pattern Subontology

### 6.4.3 Security Ontology Validation

To evaluate the usefulness of ontology, consistent answers must be given to real world questions when using it for inference. According to [162], competency questions have been defined and used to validate the proposed ontology.

In this section, a number of questions are listed which are likely involved in the project development and come up with by the developers. The questions are designed as indicative of what the ontology can handle and reason about rather than as exhaustive as possible. Each of the questions is firstly expressed formally as a DL-query, which is a query language that can be used to query RDF and OWL-DL ontologies, and then the query results are presented with comments in appropriated place. Figure 6-12 illustrates one of the query executions in Protégé Editor. Several of competency questions designed for validating the proposed security ontology are shown as follows.

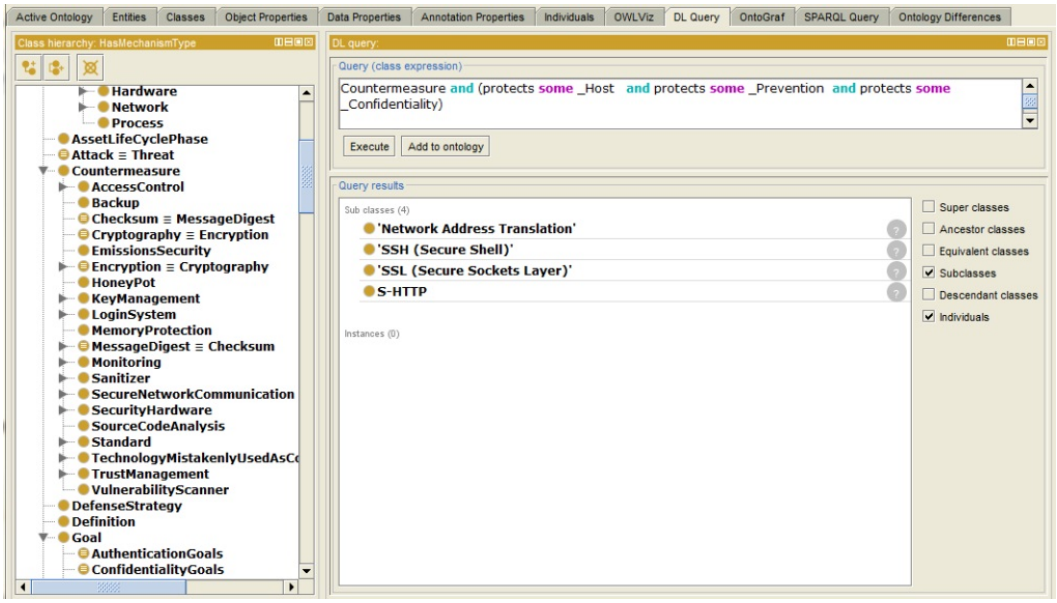



Figure 6-12 Example of Query Result in Protégé Editor

Q1: Which assets are confidential?

*DL Query: Asset and (SecurityAttribute value Confidential))*

*DL Result: Internal data*

*Sensitive data*

*Sensitive process*

Q2: Which threats threaten the integrity attribute of internal data assets in the network layer?

*DL Query: Threat and (threaten some (Asset and (Asset value InternalData) and (SecurityAttribute value Integrity)) and (resideOn value Network))*

*DL Result: Spoofing*

*Session Hijacking*

Q3: Which security patterns protect the sensitive data against network eavesdropping threat?

*DL Query: SecurityPattern and (hasProblem some (Threat and (Threat value NetworkEavesdropping) and (threaten value SensitiveData)))*

*DL Result: Secure pipe*

*Secure communication*

*Secure Association*

Q4: Which security patterns can be used in Web and J2EE domain to address the SQL injection threat?

*DL Query: SecurityPattern and (hasDomain value WebAndJ2EE) and (hasProblem and (Threat value SQLInjection))*

*DL Result: Input validator*

### 6.4.4 Ontology based Security Pattern Selecting

Security patterns are used by developers to fulfil the security requirements. In this section, a method is developed to identify and retrieve the "right" security patterns from the security pattern base to fulfil the security requirements elicited from the previous evolution stage. Then, the selected patterns will be instantiated and integrated into the system design model. Figure 6-13 depicts the pattern selecting process.

To facilitate the selection of security patterns from pattern repository, a pattern search engine is designed. For a given security requirement, pattern search engine will try to find one or more security patterns that fulfil it. Two kinds of people can be the potential

users of this search engine. One of them is the software system developer. This kind of users can input the specific request according to the security requirement or treat the whole security requirement list as the input, and then the search engine attempts to match the security requirements with the security patterns by inferring the proposed security ontology using Protégé OWL API and then search the corresponding security pattern specification from the pattern repository. The other kind of user is the security pattern developers who update the patterns and its corresponding ontology description in both security pattern repository and security ontology repository.

The patterns have been organised and classified with the proposed multiple aspects method with the consideration of dependency relationships between patterns.

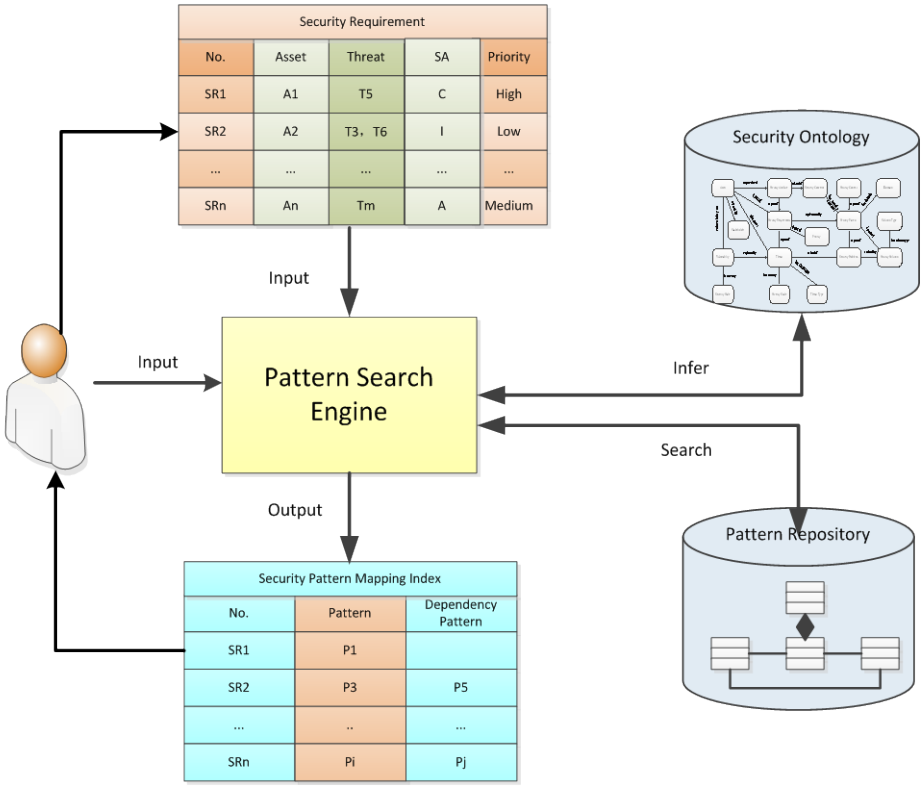



Figure 6-13 Pattern Selection Process

The pattern search engine can be implemented by incorporating OWL API and is composed of four functions:

- Input function. An input function receives the user's required security requirement or takes the set of security requirements as input.

- Infer function. An infer function infers the developed security ontology to find the security patterns according to the user input by using OWL API. The core of infer function is the algorithms realising the mapping.
- Search function. A search function will search the security pattern repository according to the mapping result of infer function and returns the development specification of the selected patterns which can be used by developer.
- Output function. An output function returns the mapping index between security requirement and mitigation security patterns.

The key part of the pattern search engine are some algorithms that match the security patterns with required security requirements until either there are no more security requirements existing, or no more security patterns which can be matched with them.

In order to extract the corresponding results from the proposed security ontology, the Protégé OWL API can be used to encode the competency questions in the algorithm structure. The OWL API is a Java application interface and reference implementation for creating, manipulating and serialising OWL Ontologies [78]. In the following, two of representative algorithms are given in a pseudo code format to show how the search engine performs the infer function.

By incorporating OWL API, Algorithm 1 is used to determine the assets threatened by a given threat with the consideration of a given stakeholder. In *GetRelated(x, y)* function, *x* is a given concept, while *y* is a relation (also called object property in Protégé OWL). *GetRelated(x, y)* function returns a collection of concepts which are related with *x* via *y*. The *GetInstances(x)* function returns a collection of instances (also called individuals in Protégé OWL) belonging to concept *x*.

**Algorithm 1** Asset Owned by a Given Stakeholder and Threatened by a Given Threat

| | |
|---|---|
| Input | *T* is the given threat<br>*S* is the given stakeholder |

| Output | *A* is the asset array |
|---|---|
| Initialisation | *A*=∅ |

procedure getAsset(*T, S*) return *A*

1. *T*← given threat
2. *S*← given stakeholder
3. *A*←Null

   {* *A* returns a collection of asset owned by *S* and threatened by *T* *}
4. *RAL*←GetRelated(*T*, sr:hasAsset)

   {*sr:hasAsset specifies that the object property "hasAsset" in security requirement subontology "sr" as the relation *}
5. for *i*← 0 to *RAL*.Length do
6. *I*← GetInstances(*RAL[i]*)

   {*Exact the instances from each of related asset classes *}
7. for *j*←0 to *I.* Length do
8. if *I[j]*.sr:ownedBy==*S* then

   {*sr:ownedBy specifies that the object property "ownedBy" in security requirement subontology "sr" as the relation *}
9. *A*.Add(*I[j]*)
10. end if
11. end for
12. end for
13. return *A*

List 6-7 Algorithm of Exacting Asset Threatened by Given Threat

Algorithm 2 is used to search the security patterns which can mitigate the threats

threatening the given asset by violating the given security attributes in a given domain.

**Algorithm 2** Security Patterns Searching

| | |
|---|---|
| Input | *A* is the given asset<br>*SA* is the given security attribute<br>*D* is the given application domain |
| Output | *SP* is the security pattern array |
| Initialisation | *SP*=∅ |

procedure getAsset(*A, SA, D*) return *SP*

1. *A* ← given asset
2. *SA*← given security attribute
3. *D*← given domain
4. *SP*←Null
5. *TL*← GetRelated(*A*, sr:isThreatedBy)
6. for *i*←0 to *TL*.Length do
7. *T*← GetInstance(*TL[i]*)
8. for *j*←0 to *T*.Length do
9. if *T[j]*.sr:hasSecurityAttribute ==*SA* then
10. *P*← GetRelated(*T[j]*,sp:isSolvedBy)
11. for *k*← 0 to *P*.Length do
12. *PI* ← GetInstance(*P[k]*)
13. for *m* ← 0 to *PI*.Length do
14. if *PI[m]*.sp:hasDomain= = *D* then
15. if *PI[m]*.sp:hasLayer = = *T[j]*.sr:residesOn then

*16.* *PR*← GetRealted(*PI[m]*, sp:requires)

{**PR* is the pattern set in which pattern is required by the exacted pattern with "require" relation in security pattern subontology *sp**}

*17.* *PS*←GetRelated(*PI[m]*, sp:isSpecialisedBy)

{**PS* is the pattern set in which pattern specifies the exacted pattern with "isSpecialisedBy" relation in security pattern subontology *sp**}

18. if *PS*.Length !=0 then
19. for *l*← to *PS*.Length do
20. *SP*.Add(*PS[l]*)
21. end for
22. else
23. *SP*.Add(*P[m]*)
24. end if
25. if *PR*.Length !=0 then
26. for *n*← to *PR*.Length do
27. *SP*.Add(*PR[n]*)
28. Line 16 to Line 27 with PR[n] for PI[m]
29. end for
30. end if
31. end if
32. end if
33. end for
34. end for
35. end if

```
36.         end for
37.     end for
38.     return T
39.     return SP
```

List 6-8 Algorithm of Exacting Security Pattern

## 6.5 Security Pattern Application

Once the mappings between security requirements and security patterns have been done, it is time to integrate the selected security patterns into the extracted UML diagrams to protect the assets which are the elements representing in diagrams against their potential threats. From the output of security pattern selecting, an index will be generated which can be used as the prerequisite of this section.

The idea is to integrate security patterns at higher level of abstraction in system design. In this phase, system class diagram and security pattern diagram can be treated as two sets of class diagram. Therefore, the task of this phase is to integrate or merge these two sets. In the set theory of mathematics, the method to merge two sets is to find the set intersections. Afterwards, a merged set can be achieved by utilising the elements from two sets with subtracting the intersection elements. The method used in this thesis to integrate system models elements with security pattern elements is inspired by the method used in set merging. Figure 6-14 depicts the process of the proposed method.

The main steps of pattern integration process are:

- Input

  This step involves (1) the system model that describe the key assets of the legacy system; (2) a set of selected security patterns. Those patterns are organised using the proposed multiple aspects approach, in a security pattern repository. The structure of the selected pattern is represented as a UML class diagram, and the behaviour of the pattern is shown as sequence diagram to facilitate the understanding and using.

- Mapping

  This step includes adding the annotation to the class where security patterns are applied and analysing the elements of the corresponding security pattern also called pattern participants so as to make the association between the elements of the selected security pattern and system model elements.

- Integration

  This operation includes adding the pattern participant classes of the pattern to the stereotype class in the system model and adjusting the relations between them.

- Output

  The security integrated model is generated after applied security pattern in the appropriate place in the system model. This UML profile provide a suitable way to define a semantic for each solution provided by security pattern and allow applying this semantics directly to developing application model's elements.

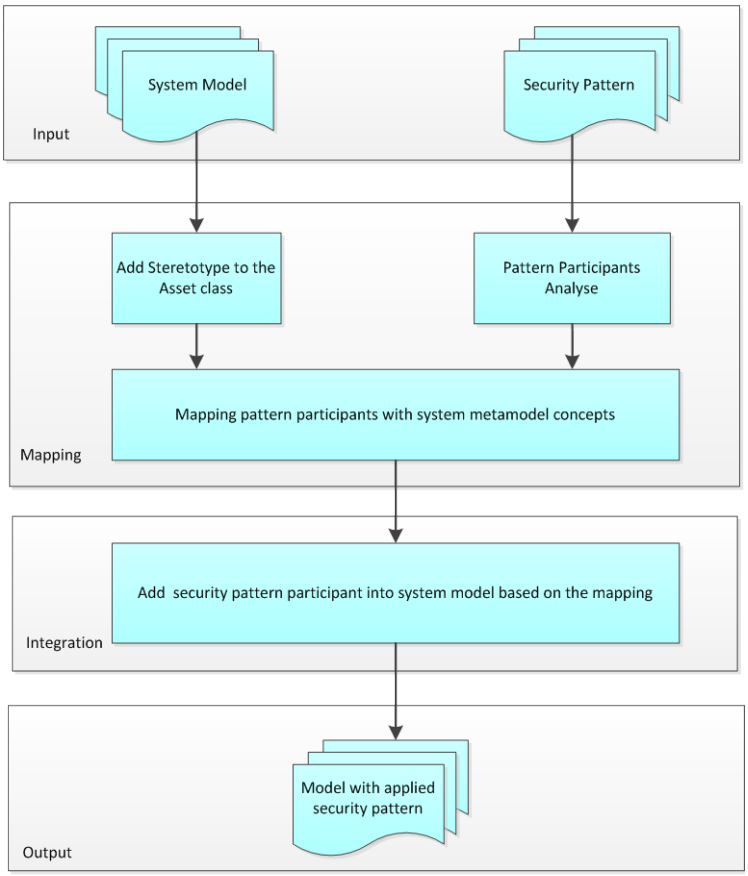



Figure 6-14 Security Pattern Integration Process

### 6.5.1 Security Annotation in System Diagram

A security annotation is an attribute or Meta that can be assigned to an element of the system diagram. In order to map the participants of the security pattern with the asset class in the system UML model, stereotype as one of the extensibility mechanisms in UML has been chosen to show where a security pattern should be applied.

### 6.5.2 Security Pattern Integration

The key to integrate the system diagram elements with security pattern participants is to find the junction between them, based on which the relationships between classes can be taken into consideration when a refined integration is analysed. The detailed considerations are listed below.

- Check whether there are some classes having the same or similar function in both parts, even if they are called different names in their own part, there still exists some degree of redundancy. Hence, the redundant class should be taken out of the combined diagram.
- Check if there exists the “Whole-Part” relationship between classes in both parts. If yes, they should be connected as Aggregation or Combination relationship.
- Check if there exists “Collaboration” relationship between classes to perform a particular function. If yes, the Dependency or Association relation is to be considered between them.
- If there are no obvious relations between them, some coordination classes could be added to implement the connection.

By combination, deletion of redundancy, creation relationships and coordination, an integrated system diagram with security pattern is realised.

Figure 6-15 shows part of class diagram of a web application. Suppose there is a security requirement that the web clerk can only do the use cases with the allocated authority, which means Authorisation pattern is suggested to be used. However, authentication is required before the authorisation is applied. Figure 6-16 depicts the Authenticator pattern in class diagram and Figure 6-17 shows the class diagram after the

Authenticator pattern is applied in a web application. According to the above, web clerk class is the junction and is judged as Subject role of the Authenticator pattern and a stereotype is added to indicate the correspondence. Figure 6-19 describes the class diagram for the example case after the Authorisation pattern shown in Figure 6-18 is applied, where web clerk is recognised as the subject of Authorisation pattern and customer is judged as object of Authorisation pattern with the stereotype added to them.

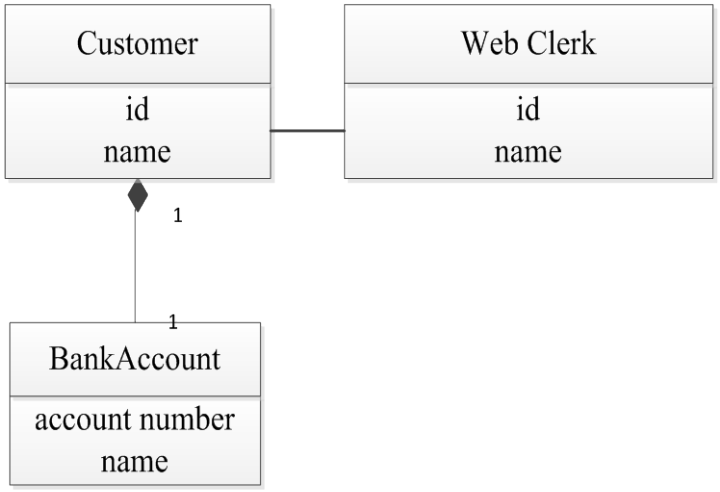



Figure 6-15 Example Class Diagram

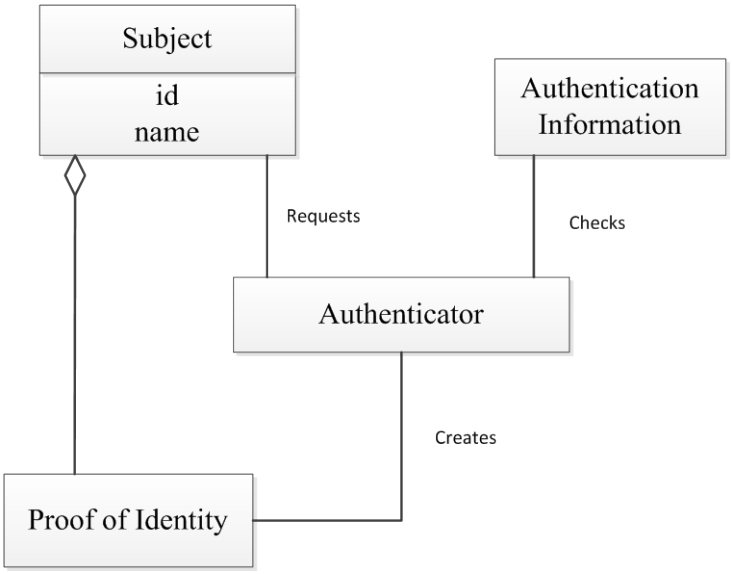



Figure 6-16 Class Diagram of the Authenticator Pattern

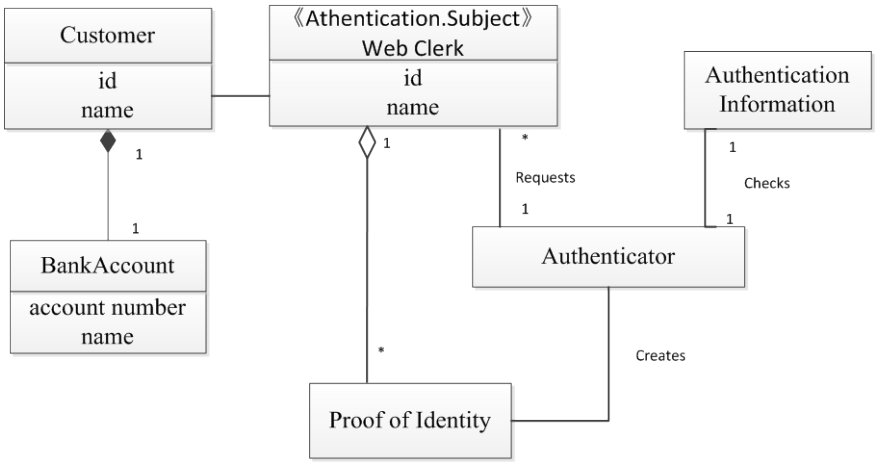



Figure 6-17 Class Diagram after Authenticator Pattern is Applied

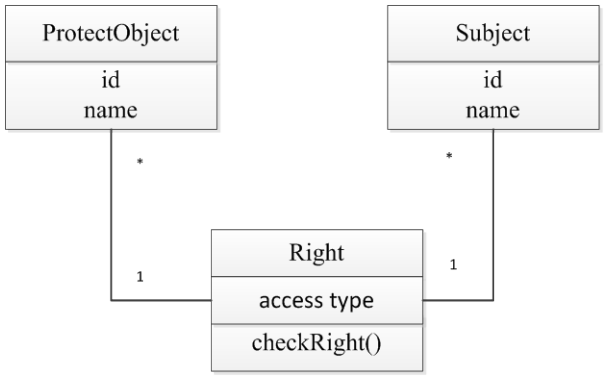



Figure 6-18 Class Diagram of the Authorisation Pattern

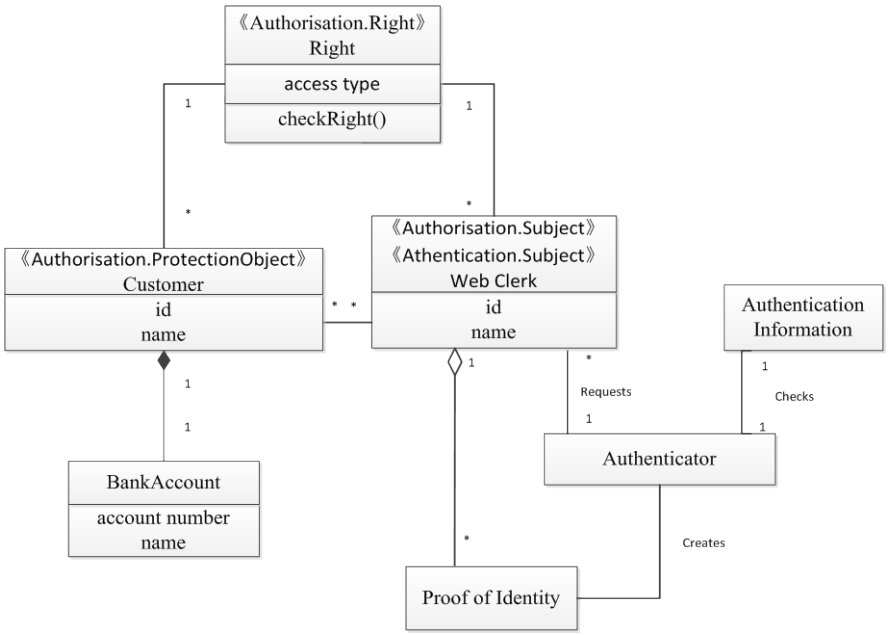



Figure 6-19 Class diagram after the Authorisation pattern is applied

## 6.6 MDA Forward Engineering

The security enhanced design models derived from this chapter are some kinds of PIMs in the form of UML class diagram. As MDA philosophy shows PIMs can be transtformed into PSMs based on a specific implementation platform and then be transformed into code with the specific programming language.

MDA forward engineering aims to deriving code from the models by using automatic tools. Presently, there are many case tools available provided by modelling vendors. Most of the transformation works on models to codes are realised by various tools by different mapping methods. However, it is only capable to generate template codes as the automatic transformation result generally. For some tools, code generation process can produce the source code by replacing the token predefined by program developer in the program skeleton.

In the following content, there is summary and catalogue for most common UML tools supporting MDA code generation till Jan. 2014.

Table 6-3 Part of UML Tools Supporting Code Generation

| Tool (Latest version) | UML Ver. | Code | XMI | Platform | License | Note |
|---|---|---|---|---|---|---|
| Acceleo (3.4) | 2 | J2EE,.Net, PHP | No | Java | Commercial | Integration with Eclipse and EMF |
| AndroMDA (3.4) | 2.2 | J2EE/EJB, Spring, Hibernate, | Yes | | Free | An open source code generation framework that follows the Model Driven |
| Apollo for Eclipse(3.0) | 2.1 | Java | No | Java | Commercial | A UML extension to Eclipse. Supporting Round-trip engineering for Java 5. |
| ArgoUML (0.34) | 1.4 | C#, Java | Yes | Java | Free | ArgoUML is the leading open source UML modelling tool and includes support for all standard |

| | | | | | | |
|---|---|---|---|---|---|---|
| Astade (1.1.2) | 2 | C++ | No | | Free | UML tool for C++ code generation |
| BOUML () | 2 | Java,C++, IDL, PHP | No | Windows Linux, MacOS | Commercial | A UML diagram designer, supporting code generation and code reverse engineering |
| Dia2code (0.8.5) | 2 | C, C++, C#, Java, | No | Linux, Windows | Free | A small utility used to generate code from a Dia diagram |
| EclipseUML (2.2) | 2.2 | JavaEE | Yes | Java | Commercial | A UML tool integrated Eclipse. It supports reverse engineering from the byte code to class and sequence diagram and forward engineering to Java code |
| Enterprise Architect (10) | 2.4 | Java, C++,C#, VB.net, XML | Yes | Windows Linux | Commercial | Full lifecycle modelling for business, software and system. It supports code generation from UML Class or Interface model |
| MagicDraw (17.0.5) | 2.2 | Java, C++, C# | Yes | Java | Commercial | Provide code engineering mechanism with full round-trip support. It provides the industry's best code engineering mechanism (with full round-trip support for Java, C++, C#, CL (MSIL) and CORBA IDL programming languages). |
| OpenAmeos (10.2) | 2 | C, C++, Java, Ada95 | Yes | Windows Linux, Windows | Free | A real-time embedded system modelling tool supporting MDA based code generation templates |
| PowerDesigner (16.5) | 2 | Java, C++, C#, VB.net, | No | Windows | Commercial | Support MDA software design, data modelling, code generation and Eclipse plugin |
| Rational Software Architect (9.0) | 2.1 | Java, C++, VB, SQL, Delphi, Oracle | Yes | Windows Linux, Unix | Commercial | An IBM powerful Product supporting integrated design, modelling and development |

| | | | | | | |
|---|---|---|---|---|---|---|
| StarUML (5.0) | 2 | Java, C++, C# | Yes | Windows | Free | UML/MDA platform |
| TopCased (5.3.1) | 2.1 | Java | No | Java | Free | UML plugin in Eclipse focusing on critical system modelling |
| Visio 2010 | 2 | C++, C#,VB | Yes | Windows | Commercial | A Microsoft diagram tool supporting UML and code generation from UML |
| Visual Paradigm (11.0) | 2 | C++, Java | Yes | Windows Linux, Mac, | Commercial | Support software development in requirement gathering, software design with UML, database design and code |

From the above tables, there are a large amount of UML tools on the market with various functions and aiming at different systems which are shown in the details specification. The available tools might not as strong as expected. However, they are supports on current research stage and can be valuable references for the future improvement on transformation.

## 6.7 Summary

In this chapter, security enhancement in software evolution is realised by adopting security patterns as security countermeasures, which incorporates proven security expert knowledge. Due to the lack of semantic, security patterns are organised by multi-aspect classification and a security search engine is designed to support the automatic matching security pattern according to the specific security requirement. The contents covered in this chapter are concluded as follows:

- A multiple aspects classification approach is proposed to organise the security patterns from the different perspectives, which are life cycle, architectural layer, application context, problem type, domain, and security concern.
- A security ontology is developed to model the concepts and relationships in security requirements and security pattern domain.
- With the proposed security ontology, a security search engine is designed to

support the matching of security requirement with the security pattern which is used to address the security needs of the software developer.

- At last, the selected patterns are instantiated and integrated into the system models with the proposed method.
- The features of the proposed security enhancement approach lie in:
  - Security patterns, as the security solution to the elicited security requirements, incorporate proven security expertise. They can be used by security novice or software developer other than the approaches in in [85] and [142] which are meant to be used by security expert.
  - Combining ontology technique with security pattern contributes to a better flexibility and reusability to the evolved system. The developed security ontology includes the concepts of risk analysis and security pattern at a higher abstraction level, than that of the approach in [142] or [127]. Therefore, it provides a more flexible way to integrate with existing approaches.
  - It can be used to manage the security patterns by annotation. Security patterns can be annotated by assigning corresponding meta-information in the proposed security ontology which will keep the patterns themselves untouched. Usually, the software developers open mapped patterns documents (e.g. HTML page) according to given security requirements, can browse through it and integrate into existing design models.

# Chapter 7

# Case Study

**Objectives**

- To illustrate how to use the related tools in the proposed approach
- To demonstrate the way of applying the overall proposed reengineering approach SEMDA to evolve software for security concerns

## 7.1 Overview

This chapter presents how the SEMDA approach is implemented in a real legacy system. There are five core techniques proposed in this thesis for security evolution: model slicing, system decomposition, security mechanism detection, security requirement elicitation, and ontology based security pattern for security improvement. In this chapter, the case study is examined by employing the proposed techniques respectively.

## 7.2 Case Study Introduction

WebStoreApp [174] is an open source online shopping application project based on GNU General Public License version 2.0 (GPLv2) which allows customers to shop brand new products as well as used items. WebStoreApp is written in Java using Struts 2.x and Hibernate 3.x frameworks. It Includes 62 classes, including 21 action classes (servlet), 12 beans classes (EJB), 12 interface classes and 1 POJO class implementation of Facade design pattern etc. It contains 10740 lines of code and provides the basic function for online shopping. Figure 7-1 and Figure 7-2 show the user account screenshot and order processing screenshot of this application respectively.

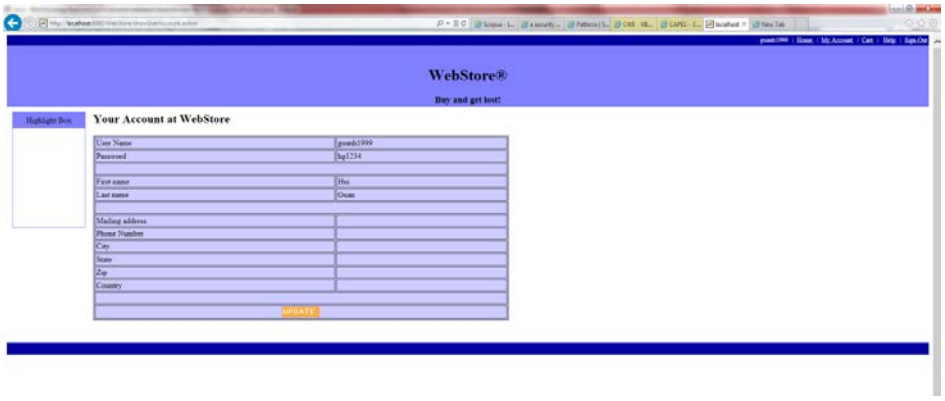



Figure 7-1 User Account Screenshot of WebStoreApp

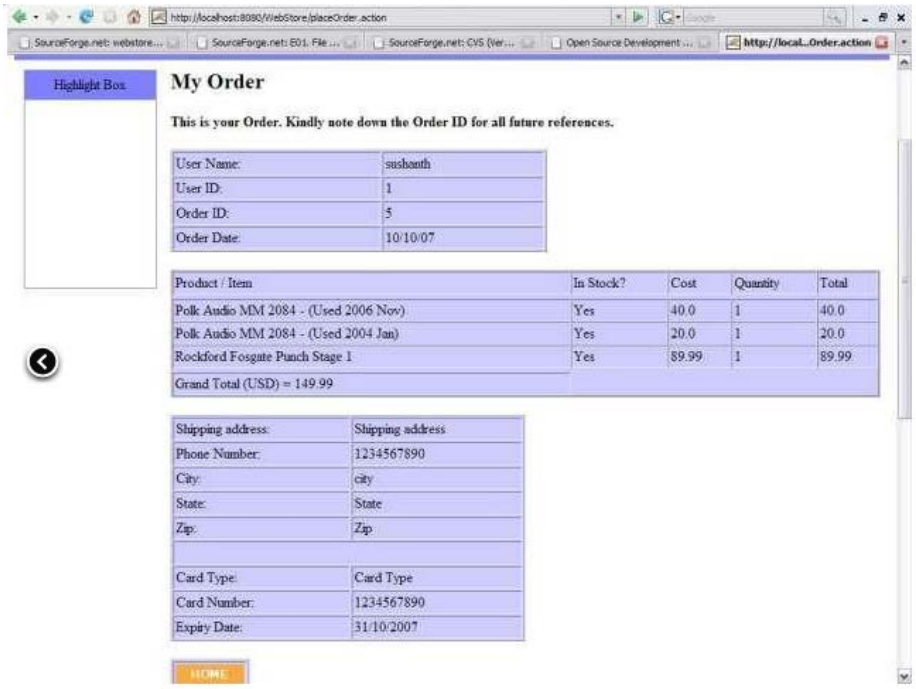



Figure 7-2 Order Processing Screenshot of WebStoreApp

## 7.3 Legacy System Understanding and Extraction

The analysis for the legacy system is to understand the current system and designing a decomposition strategy for the reusable legacy assets identification. It is crucial because legacy systems can be implemented by different building approaches and programming languages, many of them do not have clear specifications.

### 7.3.1 Architecture Recovery

Being imported into java NetBeans, the file directory structure of WebStoreApp is shown in Figure 7-3. The execution of Login function shown in Figure 7-5 is chosen as the example of analysing in order to recovery the real architecture of the WebStoreApp with the help of typical J2EE application architecture shown in Figure 7-4.

In Figure 7-4, the typical client of J2EE application is a web browser which accesses the web tier where the servlets reside. Servlets then forward the requests to EJBs (Enterprise Java Beans) which reside in the business tier. Then, some of them access the database to perform the request. Based on the analysis of Figure 7-3, Figure 7-4, the architecture of WebStoreApp is recovered and illustrated in Figure 7-6.

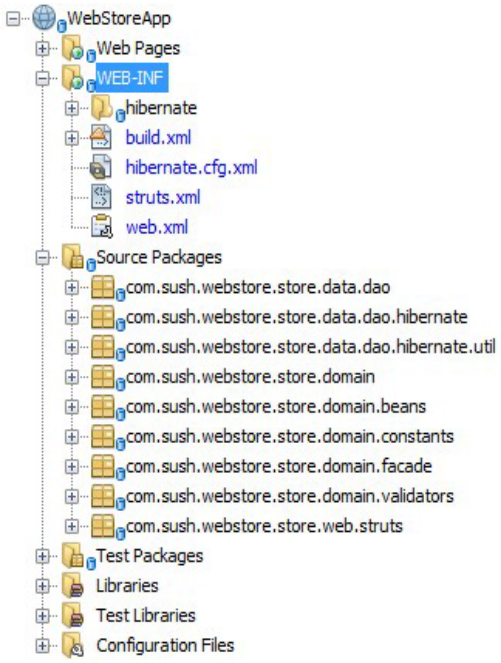



Figure 7-3 File Directory Structure of WebStoreApp

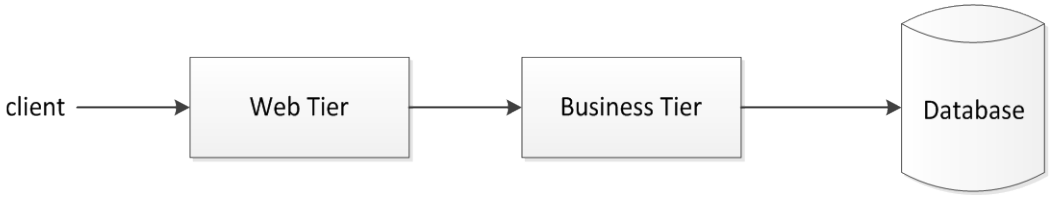



Figure 7-4 A Typical J2EE Web Application Architecture

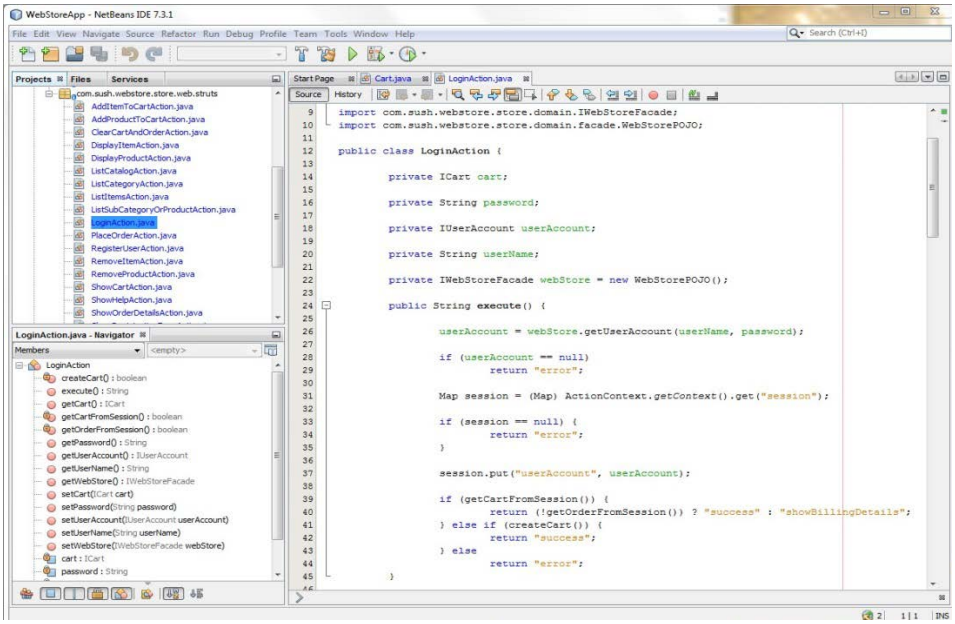



Figure 7-5 Source Code Screenshot of LoginAction Servlet

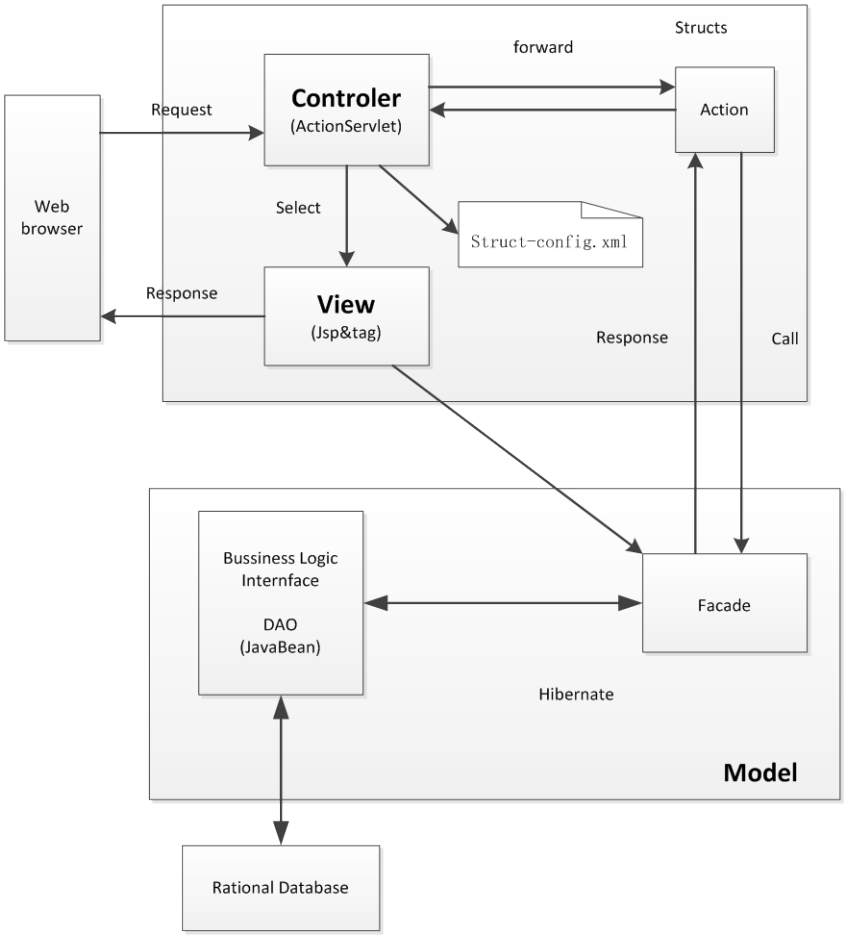



Figure 7-6 Architecture of WebStoreApp based on Struts and Hibernate

WebStoreApp was developed using Hibernate and Struts techniques building on MVC (Model-View-Controller) pattern. Model-View-Controller (MVC) is a popular architecture to separate concerns in a software development. The Model represents the business or database code, the View represents the page design code, and the Controller represents the navigational code. The Struts framework is designed to help developers create web applications that utilise MVC architecture. Hibernate is an open source framework used to manage the persistent data from Java environment to database.

Facade pattern is used in WebStoreApp design which hides the complexities of the system and provides an interface to the client using which the client can access the system. It often makes sense to have a higher-level interface for a model, called the facade.

### 7.3.2 System Model Extraction

Visual Paradigm for UML (VP-UML) [169] is a powerful UML CASE Tool from the OMG. It can support modelling of UML 2, Business Process Modelling Notation (BPMN) and SysML. Moreover, it provides reverse engineering and forward engineering for java, C# programming language. Class diagram and sequence diagram generation for java is provided and what's more, code generation from class diagram is supported as well.

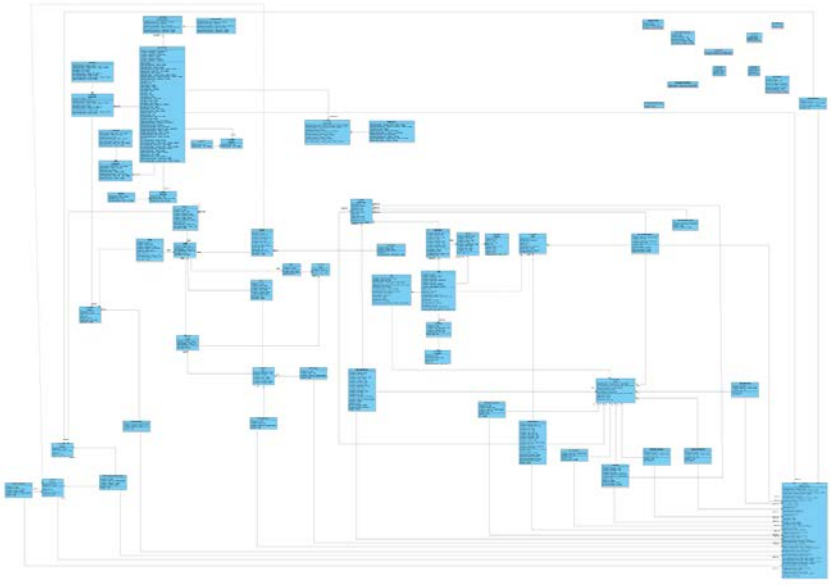


Figure 7-7 Entire Class Diagram of WebStoreApp

In this section, VP-UML is used to generate class diagram and sequence diagram from the source code of WebStoreApp. Figure 7-7 shows the entire class diagram of WebStoreApp, and all of the bean classes and their relationship are shown in Figure 7-8. Several extracted sequence diagrams are depicted in Figure 7-9, Figure 7-10 and Figure 7-11.

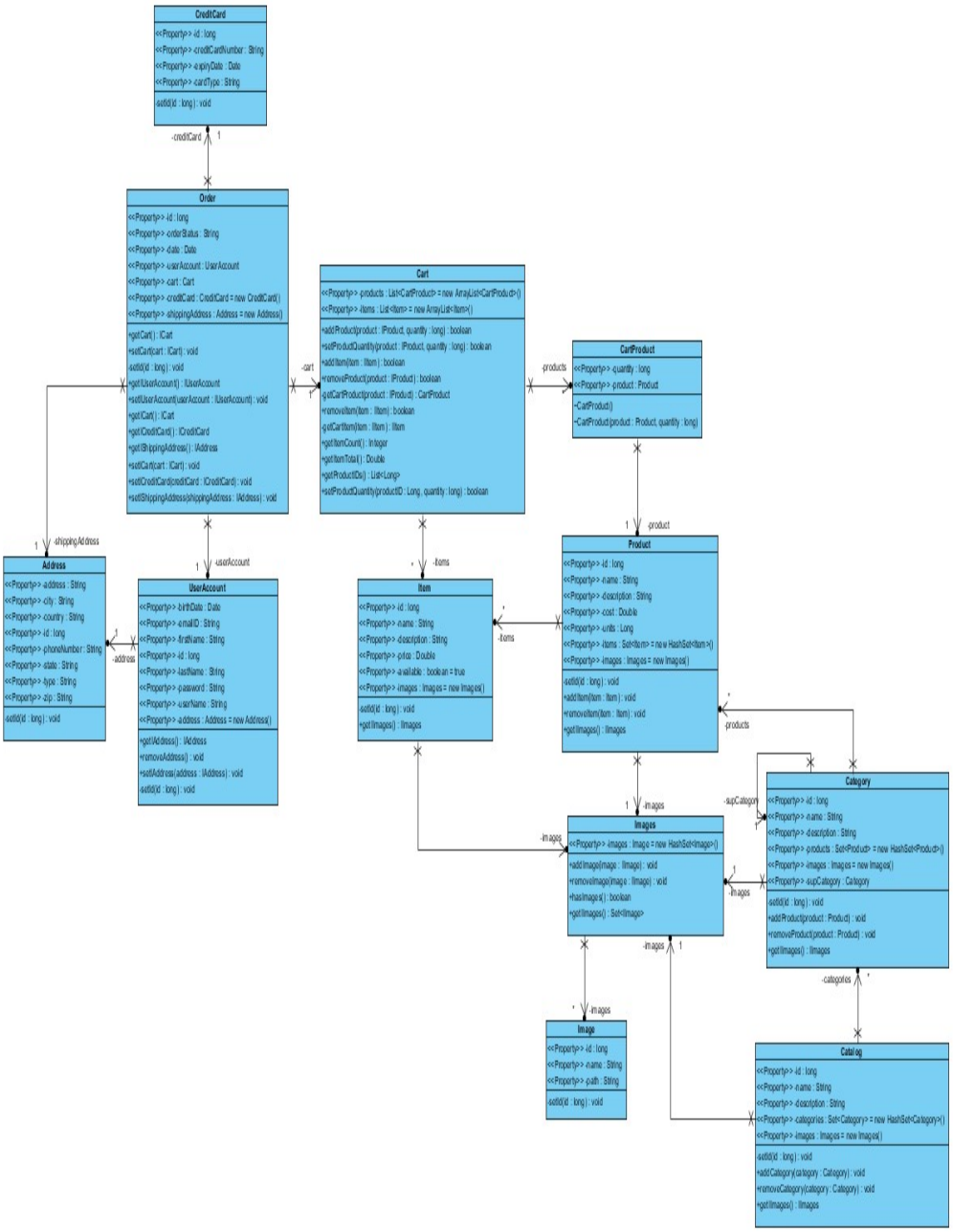



Figure 7-8 Class Diagram of WebStoreApp Bean Classes

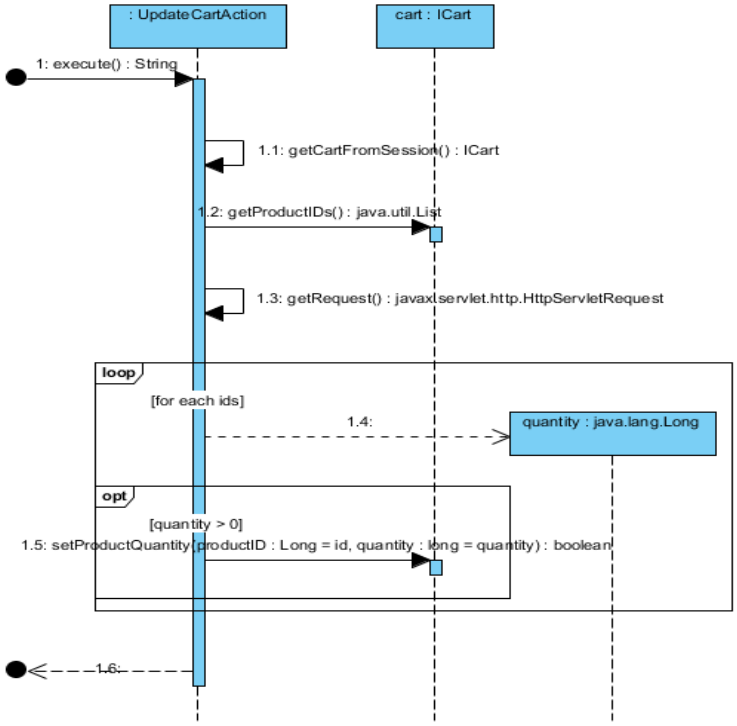



Figure 7-9 Sequence Diagram of Cart Update Operation

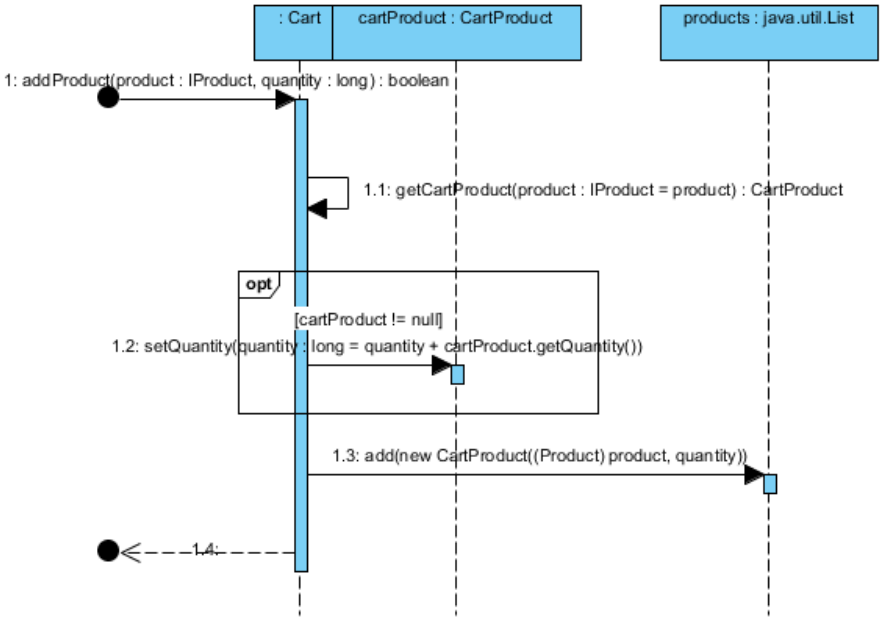



Figure 7-10 Sequence Diagram of Add Product to Cart Operation

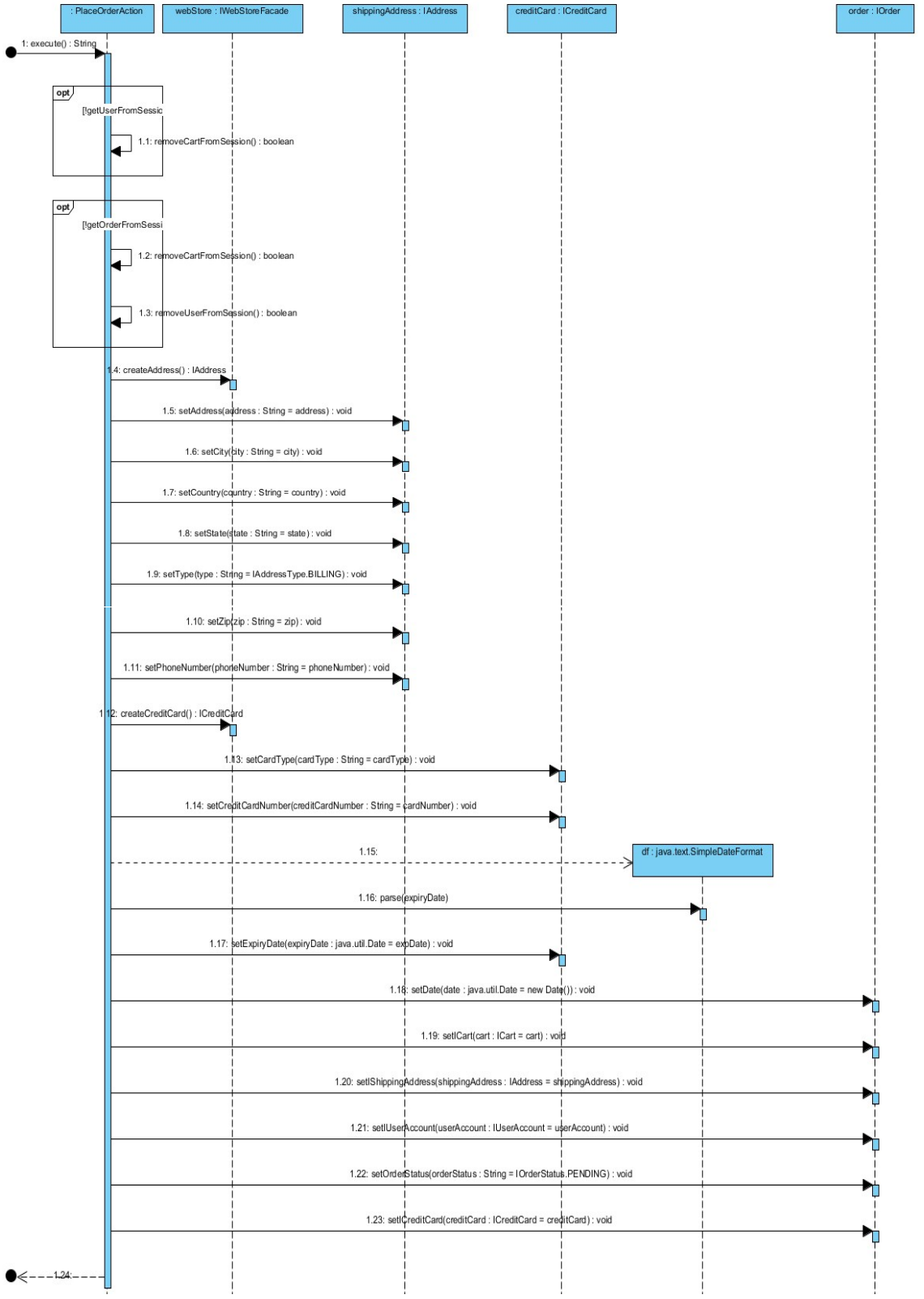



Figure 7-11 Sequence Diagram of PlaceOrder Operation

### 7.3.3 CSDG Construction of WebStoreApp

This section a CSDG is constructed based on the extracted system diagrams according to the algorithm described in Section 4.4. Figure 7-8, Figure 7-9, and Figure 7-10 are selected as the source graph to construct CSDG. In view of readability and complexity of whole system, only parts of them are shown in Figure 7-12.

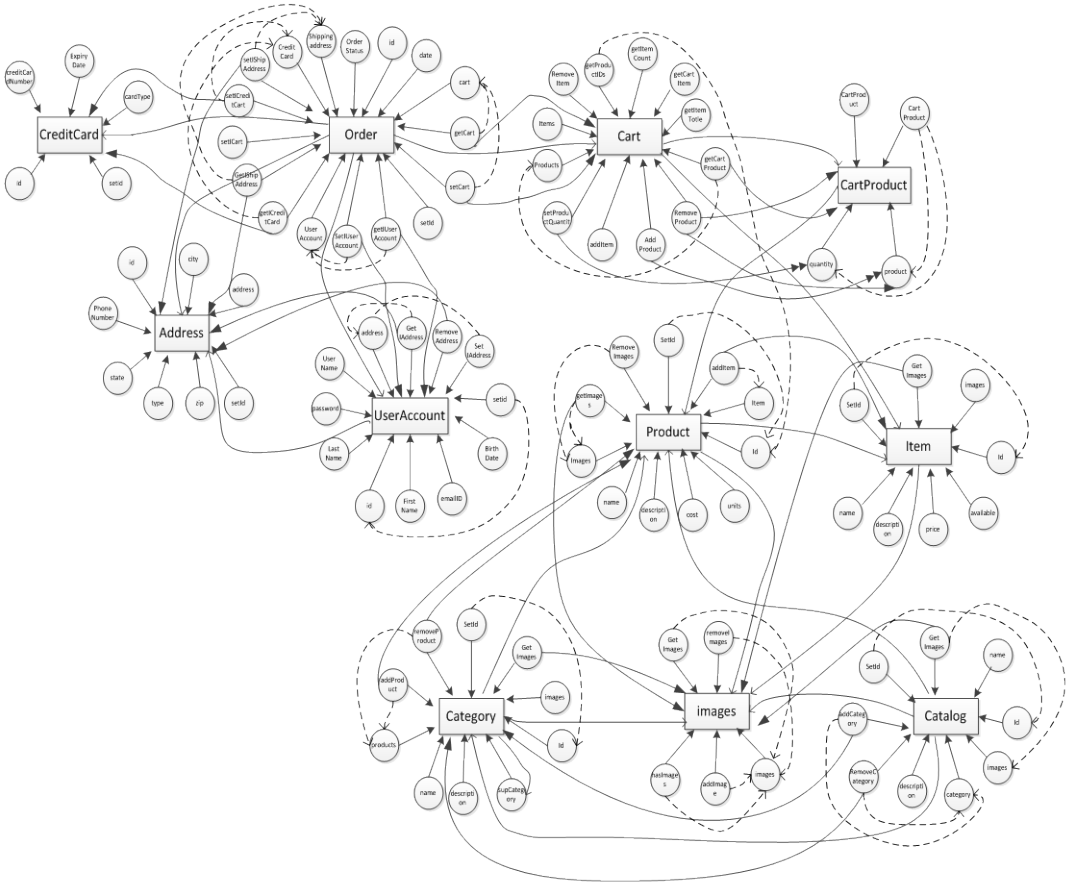



Figure 7-12 Part of CSDG for WebStore Application

Using the model slicing algorithm in Section 4.4.2, Figure 7-12 is sliced based on the given slicing criteria and the result is shown in Figure 7-13.

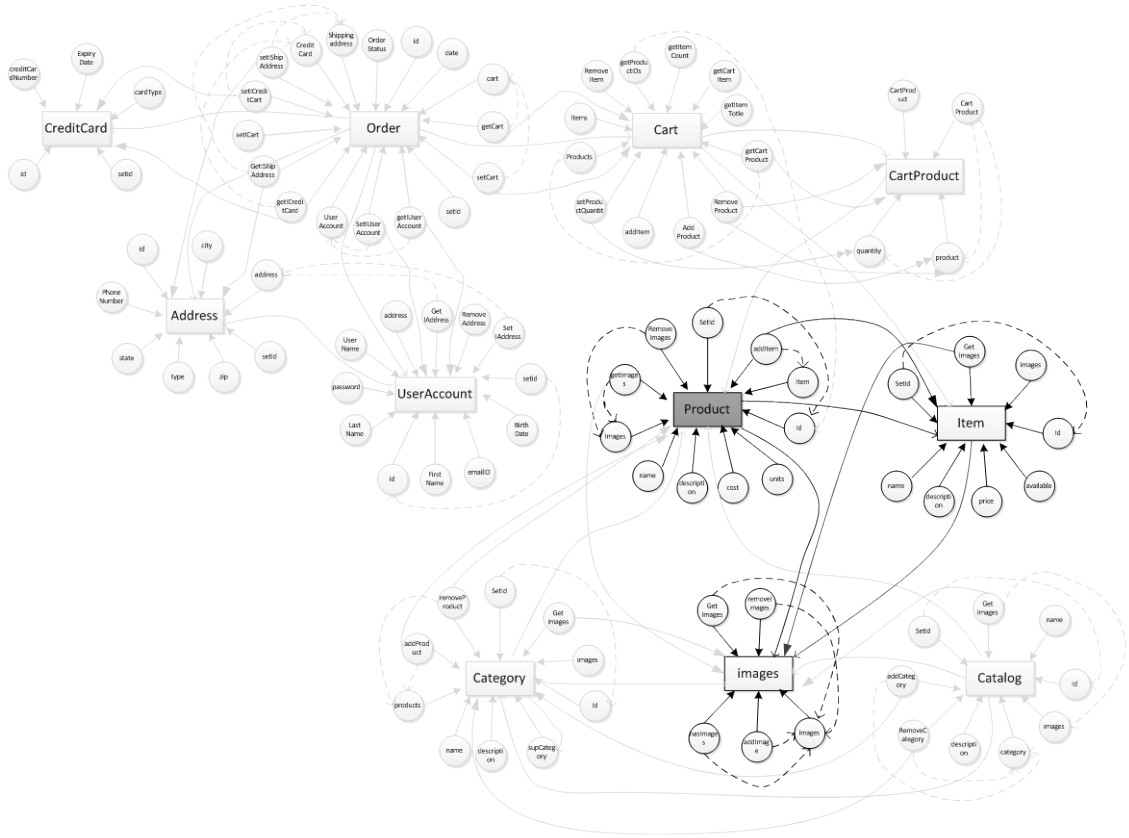



Figure 7-13 Slicing Output using Product as the Slicing Criteria

### 7.3.4 Legacy System Partition

According to the algorithm of system partition proposed in Section 4.5, the Independence Metrics are computed and the result is shown in Table 7-1.

Table 7-1 Results of One Iteration for WebStoreApp

| Cluster No. | Slice Criteria | Slice NoteSet | IM |
|---|---|---|---|
| 1 | UserAccount | UserAccount, Address | 1.6 |
| 2 | Address | Address | 0 |
| 3 | Order | Order, CreditCard, Cart, UserAccount, | 0.55 |
| 4 | CreditCard | CreditCard | 0 |
| 5 | Cart | Cart, CartProduct, Product, Item, Images | 1.2 |
| 6 | CartProduct | CartProduct, Product, Item, Images | 1.75 |

| | | | |
|---|---|---|---|
| 7 | Product | Product, Item, Images | 2.33 |
| 8 | Item | Item, Images | 2.15 |
| 9 | Catalog | Catalog, Images, Product, Item, Category | 0.36 |
| 10 | Category | Category, Product, Item, Images | 1.13 |
| 11 | Images | Images | 0 |

After iteration computing for several times, the final partition result is shown in Table 7-2.

Table 7-2 Partition Result of WebStoreApp Application

| Cluster No. | Class name |
|---|---|
| 1 | Product, Item, Images |
| 2 | UserAccount, Address |
| 3 | Cart, CartProduct |
| 4 | Order, CreditCard |
| 5 | Category, Catalog |

From the result of system partition, it can be concluded that there are five main modules in WebStoreApp which can be described as a system use case diagram in Figure 7-14.

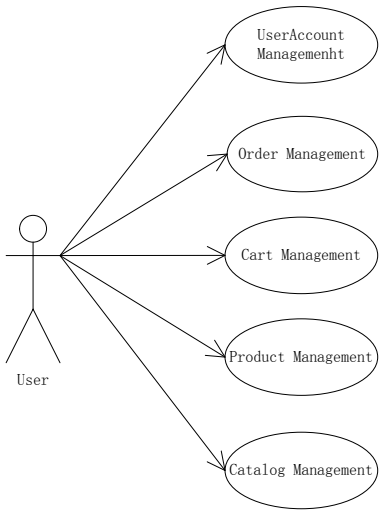



Figure 7-14 System Use Case Diagram of WebStoreApp Application

### 7.3.5 Security Mechanism Detection

According to the checklist listed in Table 4-8 in Section 4.6.3, the detected security mechanisms in the current design in WebStoreApp are:

- User name and password as authentication
- Client-Certificate

## 7.4 Security Requirements Analysis

According to the Chapter 5, security requirement is represented by the assets, the threats threatening to them and the priority representing the development order when it is to be satisfied.

### 7.4.1 Asset Analysis

By using the approach proposed in Section 5.2.1, assets in WebStoreApp are identified and their criticalities are evaluated according to Table 5-2. Table 7-3 shows the final result and the average criticality value is appended to the bottom of the table.

Table 7-3 Part of Asset Criticality Analysis for WebStoreApp

| ID | Asset Name | Asset Type | Sensitive | Criticality | Rating |
|---|---|---|---|---|---|
| 1 | UserAccount | Customer data | Yes | Very high | 10 |
| 2 | Address | Customer data | Yes | Very high | 10 |
| 3 | CreditCard | Financial data | Yes | Very high | 10 |
| 4 | Order | Customer data | Yes | Very high | 10 |
| 5 | Cart | Public data | No | Low | 3.99 |
| 6 | CartProduct | Public data | No | Low | 3.99 |
| 7 | Product | Public data | No | Low | 3.99 |
| 8 | Item | Public data | No | Low | 3.99 |
| 9 | Category | Public data | No | Low | 3.99 |
| 10 | Catalog | Public data | No | Low | 3.99 |

| | | | | | |
|---|---|---|---|---|---|
| 11 | User Registration | Business process | Yes | Very high | 10 |
| 12 | User Logging | Business process | Yes | Very high | 10 |
| 13 | Place Order | Business process | Yes | Very high | 10 |
| 14 | Clear Cart and Order | Business process | No | Low | 3.99 |
| 15 | Add Product to Cart | Business process | No | Low | 3.99 |
| 16 | Display Item | Business process | No | Low | 3.99 |
| 17 | Display Product | Business process | No | Low | 3.99 |
| 18 | List Catalog | Business process | No | Low | 3.99 |
| 19 | List Category | Business process | No | Low | 3.99 |
| 20 | List Items | Business process | No | Low | 3.99 |
| 21 | List SubCategory or Product | Business process | No | Low | 3.99 |
| 22 | Remove Item | Business process | No | Low | 3.99 |
| 23 | Remove Product | Business process | No | Low | 3.99 |
| 24 | Show Cart | Business process | No | Low | 3.99 |
| 25 | Show help | Business process | No | Low | 3.99 |
| 26 | Show Order Detail | Business process | Yes | Very high | 10 |
| 27 | Show Registration Form | Business process | Yes | Very high | 10 |
| 28 | Show User Account | Business process | Yes | Very high | 10 |
| 29 | Sign Out | Business process | Yes | Very high | 10 |
| 30 | Update Cart | Business process | No | Low | 3.99 |
| The average asset scale value | | | | | 6.19 |

### 7.4.2 Threat Analysis

To design a secure system, the possible threats threatening the system are needed to be understood. Without this understanding a system that is more expensive than necessary and that has a large performance overhead may be produced [45].

It is necessary to rate the security requirements CIAA of target application when using the proposed EDTEM in Chapter 5. Using internal guidelines based on documents such as [30], the following security objectives for the WebStoreApp application may be produced:

- Confidentiality: High

Application is ready for use to unknown public users, and sensitive financial data and client private information are handled by this application.

- Integrity: Medium

Loss of integrity will have an adverse effect on confidentiality. Poor inventory and shopping tracking may result in significant financial loss to the company and may result in customer dissatisfaction / loss of customers.

- Availability: Low

Information must be readily available with flexible tolerance for delay, or loss of availability will have an adverse effect. A major disruption of the application will cause a delay in shopping and have some financial consequences to the organisation.

- Accountability: Medium

Operations to sensitive and internal data should be identified and that the trace to the author and the operation is kept.

According to the EDTED described in the previous section, almost all of the likely threats are included in this type of web applications due to their complicated environment and target user.

The threat list can be further screened out according to the security requirements of CIAA aspects. In terms of the EDTEM algorithm, threats with only Availability requirements can be rule out in that the given application has low requirements, while threats with Confidentiality (C), Integrity (I) and Accountability (Ac) are remained.

Table 7-4 shows the threat list threatens the asset of WebStoreApp application.

Table 7-4 Threat List Threatens the Asset of WebStoreApp

| Asset Type | Application Type | CIAA Requirement | Threat List with Threat ID |
|---|---|---|---|
| Internal Data | WA4 | Confidentiality Integrity Accountability | T12,T13,T14,T15,T16,T17,T18,T19, T20,T21,T22,T23,T24,T25,T26,T27, T29,T30,T31,T32,T33,T34,T35,T36, T37,T38,T39,T40,T41,T42,T43,T45 |
| Public data | WA4 | Integrity | T3,T4,T10,T11,T12,T13,T14,T15,T20, T21,T22,T24,T25,T26,T29,T31,T32,T 34,T35,T41,T42 |
| Sensitive data | WA4 | Confidentiality Integrity Accountability | T12,T13,T14,T15,T16,T17,T18,T19, T20,T21,T22,T23,T24,T25,T26,T27, T29,T30,T31,T32,T33,T34,T35,T36, T37,T38,T39,T40,T41,T42,T43,T45 |
| Sensitive process | WA4 | Confidentiality Accountability | T16,T17,T19,T20,T21,T22,T24, T25, T26,T27,T29,T30,T31,T32,T34, T35, T36,T37,T38,T39,T40,T41,T42, T45 |

As described in Section 5.2.2, DREAD is used to quantify the security level for each threat identified from the proposed EDTEM approach according to the rating value in Table 5-8, the result for WebStoreApp is shown in Table 7-5. Moreover, each threat is classified based on the STRIDE [160] scheme described in Section 6.3.3 for later use.

Table 7-5 Threat Risk Quantification

| AHN | ID | Threat Name | Type | D | R | E | A | D | Total |
|---|---|---|---|---|---|---|---|---|---|
| Network | 1 | Information Gathering | I | 0 | 5 | 5 | 0 | 5 | 3 |
| | 2 | Sniffing | I | 5 | 5 | 5 | 0 | 5 | 4 |
| | 3 | Spoofing | S | 5 | 5 | 0 | 5 | 5 | 4 |
| | 4 | Session Hijacking | S | 5 | 5 | 5 | 5 | 5 | 5 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 5 | Denial of Service | D | 5 | 5 | 5 | 10 | 5 | 6 |
| Host | | 6 | Viruses, Trojan horses, and Worms | STIE | 10 | 10 | 10 | 10 | 10 | 10 |
| | | 7 | Footprinting | S | 5 | 5 | 5 | 5 | 0 | 4 |
| | | 8 | Password Cracking | I | 10 | 5 | 5 | 5 | 5 | 7 |
| | | 9 | Denial of Service | D | 5 | 5 | 5 | 10 | 5 | 6 |
| | | 10 | Arbitrary Code Execution | TI | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 11 | Unauthorised Access | TI | 10 | 5 | 5 | 5 | 0 | 5 |
| Application | In put Validation | 12 | Buffer Overflow | T | 10 | 5 | 5 | 5 | 5 | 5 |
| | | 13 | Cross-Site Scripting (XSS) | T | 10 | 5 | 5 | 5 | 10 | 7 |
| | | 14 | SQL Injection | T | 10 | 5 | 5 | 5 | 10 | 7 |
| | | 15 | Canonicalisation | T | 10 | 5 | 0 | 0 | 10 | 5 |
| | Authentication | 16 | Network Eavesdropping | I | 10 | 5 | 5 | 10 | 0 | 6 |
| | | 17 | Brute Force Attacks | I | 5 | 0 | 5 | 5 | 5 | 4 |
| | | 18 | Dictionary Attacks | I | 5 | 0 | 5 | 5 | 5 | 4 |
| | | 19 | Cookie Replay | S | 5 | 0 | 0 | 5 | 0 | 2 |
| | | 20 | Credential Theft | S | 10 | 0 | 5 | 5 | 5 | 5 |
| | Authorisation | 21 | Elevation of Privilege | E | 10 | 10 | 0 | 5 | 0 | 5 |
| | | 22 | Disclosure of Confidential Data | I | 5 | 10 | 5 | 5 | 5 | 6 |
| | | 23 | Data Tampering | T | 10 | 5 | 5 | 5 | 5 | 6 |
| | | 24 | Luring Attacks | E | 10 | 5 | 0 | 5 | 5 | 5 |
| | Configuration Management | 25 | Unauthorised Access to Aministration Interface | TI | 0 | 5 | 5 | 10 | 0 | 4 |
| | | 26 | Unauthorised Access to Configuration Stores | TI | 0 | 5 | 5 | 10 | 0 | 4 |
| | | 27 | Retrieval of Clear Text Configuration Data | I | 0 | 5 | 5 | 10 | 0 | 4 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 28 | Lack of Individual Accountability | R | 0 | 5 | 5 | 10 | 0 | 4 |
| | | 29 | Over-Privileged Process and Service Accounts | E | 0 | 5 | 5 | 10 | 0 | 4 |
| | Sensitive Data | 30 | Access sensitive data in storage | I | 10 | 5 | 0 | 10 | 5 | 6 |
| | | 31 | Network Eavesdropping | I | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 32 | Data Tampering | T | 10 | 5 | 5 | 5 | 5 | 6 |
| | Session Management | 33 | Session Hijacking | S | 5 | 5 | 5 | 5 | 10 | 6 |
| | | 34 | Session Replay | S | 5 | 5 | 5 | 5 | 10 | 6 |
| | | 35 | Man in the Middle | TI | 5 | 5 | 5 | 5 | 5 | 5 |
| | Cryptography | 36 | Poor Key Generation or Key Management | I | 5 | 5 | 5 | 0 | 0 | 3 |
| | | 37 | Weak or Custom Encryption | I | 5 | 5 | 5 | 0 | 0 | 3 |
| | | 38 | Checksum Spoofing | I | 5 | 5 | 5 | 0 | 0 | 3 |
| | Parameter Manipulation | 39 | Query String Manipulation | I | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 40 | Form Field Manipulation | I | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 41 | Cookie Manipulation | TI | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 42 | HTTP Header Manipulation | T | 5 | 5 | 5 | 5 | 5 | 5 |
| | Exception | 43 | Attacker Reveals Implementation Details | I | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 44 | Denial of Service | D | 5 | 5 | 5 | 10 | 5 | 6 |
| | Auditing and logging | 45 | User Denies Performing an Operation | R | 5 | 10 | 5 | 5 | 5 | 6 |
| | | 46 | Attacker Exploits an Application Without Trace | R | 5 | 0 | 5 | 5 | 5 | 4 |
| | | 47 | Attacker Covers His or Her Tracks | R | 5 | 0 | 5 | 5 | 5 | 4 |

In many cases, it is impossible to mitigate every threat, even if it could be done. It would almost certainly take place at the cost of decreased usability. Due to the

application budget and time, it is not necessary to mitigate every threat to the system. Therefore, some threats with more DREAD quantification value are chosen as the detailed security requirements to be alleviated. In this thesis, the threats with DREAD value lager than 5 are chosen for further processing.

Table 7-6 Asset versus Threat List with DREAD Value >5

| Asset Type | CIAA Requirement | Threat List | | |
|---|---|---|---|---|
| | | Threat | AHN | STRIDE |
| Internal data | Confidentiality Integrity Accountability | T13 | A | Tampering |
| | | T14 | A | Tampering |
| | | T16 | A | Information Disclosure |
| | | T22 | A | Information Disclosure |
| | | T23 | A | Tampering |
| | | T30 | A | Information Disclosure |
| | | T32 | A | Tampering |
| | | T45 | A | Repudiation |
| Public data | Integrity | T13 | A | Tampering |
| | | T14 | A | Tampering |
| | | T23 | A | Tampering |
| Sensitive data | Confidentiality Integrity Accountability | T13 | A | Tampering |
| | | T14 | A | Tampering |
| | | T16 | A | Information Disclosure |
| | | T22 | A | Information Disclosure |
| | | T23 | A | Tampering |
| | | T30 | A | Information Disclosure |
| | | T32 | A | Tampering |
| | | T33 | A | Spoofing |
| | | T34 | A | Spoofing |
| | | T45 | A | Repudiation |

| | | | | |
|---|---|---|---|---|
| Sensitive process | Confidentiality<br>Accountability | T16 | A | Information Disclosure |
| | | T22 | A | Information Disclosure |
| | | T30 | A | Information Disclosure |
| | | T45 | A | Repudiation |

The total value after DREAD threat quantification is computed by averaging all of the potential threat, the final result is 7.08.

### 7.4.3 Vulnerability Analysis

As described in Section 5.2.3, the vulnerability scanning tool named N-Stalker is used to detect the potential vulnerabilities in WebStoreApp application. Figure 7-15 depicts the scan result of vulnerability detecting for WebStoreApp application.

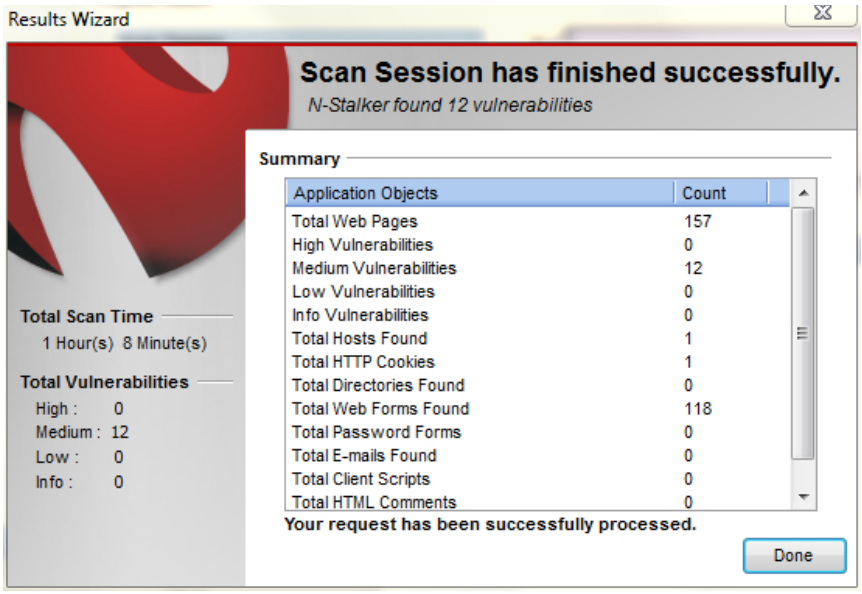
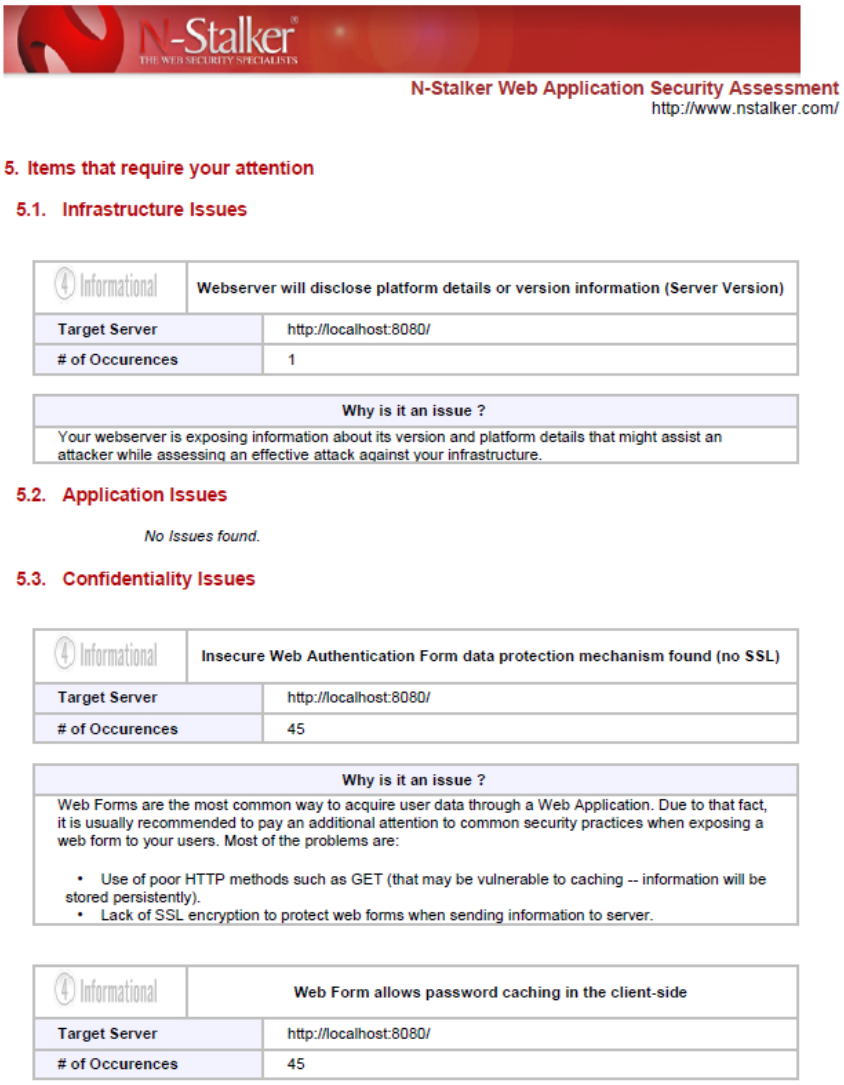



Figure 7-15 Scan Result of N-Stalker for WebStoreApp Application

Figure 7-16 shows one of the identified vulnerability of WebStoreApp application named Insecure Communications which means applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications and which is a "2007 top 10 CVE (Common Vulnerability Exploit)". The scan result is shown in Table 7-7 and CVSS is used to quantify the result depicted in Figure 7-17 and Table 7-8.


N-Stalker
THE WEB SECURITY SPECIALISTS

N-Stalker Web Application Security Assessment
http://www.nstalker.com/

5. Items that require your attention

5.1. Infrastructure Issues

| Informational | Webserver will disclose platform details or version information (Server Version) |
|---|---|
| Target Server | http://localhost:8080/ |
| # of Occurences | 1 |

| Why is it an issue ? |
|---|
| Your webserver is exposing information about its version and platform details that might assist an attacker while assessing an effective attack against your infrastructure. |

5.2. Application Issues

*No Issues found.*

5.3. Confidentiality Issues

| Informational | Insecure Web Authentication Form data protection mechanism found (no SSL) |
|---|---|
| Target Server | http://localhost:8080/ |
| # of Occurences | 45 |

| Why is it an issue ? |
|---|
| Web Forms are the most common way to acquire user data through a Web Application. Due to that fact, it is usually recommended to pay an additional attention to common security practices when exposing a web form to your users. Most of the problems are:<br>• Use of poor HTTP methods such as GET (that may be vulnerable to caching -- information will be stored persistently).<br>• Lack of SSL encryption to protect web forms when sending information to server. |

| Informational | Web Form allows password caching in the client-side |
|---|---|
| Target Server | http://localhost:8080/ |
| # of Occurences | 45 |

Figure 7-16 Security Vulnerability Detected by Using N-Stalker

Table 7-7 Vulnerabilities of WebStoreApp

| No. | Scan Policy | Vulnerability Name | Occurrences | Status | Remark |
|---|---|---|---|---|---|
| 1 | OWASP | Web server will disclose platform details or versions information | 1 | informational | |
| 2 | OWASP | Insecure web authentication from data protection mechanism found (no SSL) | 45 | informational | OWASP Top 10 version 2007 |

| | | | | | |
|---|---|---|---|---|---|
| 3 | OWASP | Web form allows password caching in the client-side | 45 | informational | |
| 4 | Full XSS Assessment | Possible Cross-Site Scripting and/or HTML injection found | 12 | Medium | OWASP Top 10 version 2010 |

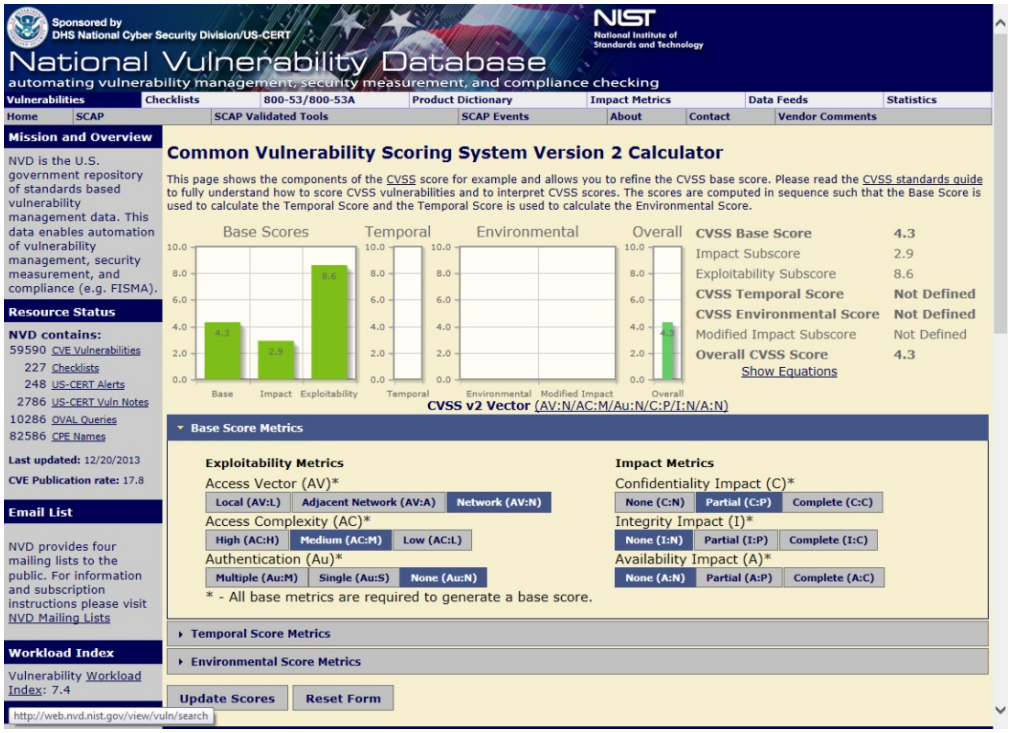



Figure 7-17 CVSS Calculator in [116]

Table 7-8 CVSS Scoring

| Vulner-ability | CVSS Base Score | | | | | | |
|---|---|---|---|---|---|---|---|
| | Access Vector | Access Complexity | Authenti-cation | Confidentiality Impact | Integrity Impact | Availability Impact | Score |
| V1 | Network | Low | None | Partial | None | None | 5 |
| V2 | Network | Medium | None | Partial | Partial | None | 5.8 |
| V3 | Network | Medium | None | Partial | None | None | 4.3 |
| V4 | Network | Medium | None | Partial | None | None | 4.3 |

The total score for vulnerabilities combined with their occurrences is: 5*1/103+5.8*45/103+4.3*45/103+4.3*12/103=4.96.

According to Formula (5) proposed in Section 5.2, the security risk of WebStore application can be computed by referencing Table 7-3, Table 7-5, Table 7-6, and Table 7-8. The value for asset is 6.19, 7.08 for threats and 4.96 for vulnerability. The final result is 6.14.

### 7.4.4 Security Evaluation

Section 5.3 described the security evaluation process with the consideration of security risk, security objectives and detected security mechanisms. Security risk is quantified by computing the threats in Table 5-7, Table 7-5 and Table 7-6 according to their DREAD value and CIAA requirements. Table 7-9 shows the evaluation result from which it can be concluded that the current security implementation in the WebStoreApp application is not enough to protect the target system to satisfy the user's security requirements. Therefore, it is necessary to evolve the target system to satisfy the required security goals.

Table 7-9 Security Evaluation for WebStoreApp Application

| | Security Objectives | Security Risk | Detected Security Mechanisms | Satisfied? |
|---|---|---|---|---|
| Confidentiality | High | Medium | Low | No |
| Integrity | Medium | High | Low | No |
| Availability | Low | Low | Low | Yes |
| Accountability | Medium | Low | Low | No |

## 7.5 Security Enhancement

As described in Chapter 6, security pattern as an encapsulation of security expert knowledge is used to improve the security level in software evolution. A security ontology is developed to implement the mapping from the elicited security requirements

to the mitigation mechanisms--security patterns which are integrated into the system diagrams afterwards.

### 7.5.1 Security Pattern Mapping

Security patterns matched given query have been found by using the developed security ontology. An example of how the ontology works with the algorithms 2 in Section 6.5 are given as follows.

For a security requirements<*Internal data, Cross-Site Scripting (XSS), Confidentiality, H*>, which means Internal data facing Cross-site Scripting threats threatening the Confidentiality security feature with High level risk, a query is designed to generate the appropriate security pattern satisfying it.

*DL Query: SecurityPattern and (hasDomain value WebAndJ2EE) and (hasLifeCycel value Design) and (hasSecurityConcerns value Confidentiality) and (has Architecture value Application) and (hasSecurityProblem and (SecurityProblem value CrossSiteScript) and (Threaten some InternalData))*

*DL Result: Intercepting Validator*

*Secure Action Base (requires relation)*

The ontology shows the query result is Intercepting Validator pattern. Secure Action Base pattern is inferred and given as well via the require relationship between them. Table 7-10 shows the mapping result with the same method as illustrated above.

Table 7-10 Mapping Result by Using the Proposed Security Ontology

| Security Requirements | | | | | | Security Pattern ID |
|---|---|---|---|---|---|---|
| Asset | | Threat | | | Priority Level | |
| Asset Type | Security | Threat | AHN | Security | | |
| Internal data | Confidentiality Integrity Accountability | T13 | A | Confidentiality Integrity | H | P13, P22 |
| | | T14 | A | Confidentiality Integrity | H | P13, P22 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | T16 | A | Confidentiality | M | P25 |
| | | T22 | A | Confidentiality | M | P3, P22 |
| | | T23 | A | Integrity | H | P3, P22 |
| | | T35 | N | Confidentiality Integrity | H | P25 |
| | | T45 | A | Accountability | M | P22, P24 |
| Public data | Integrity | T13 | A | Confidentiality Integrity | H | P13, P22 |
| | | T14 | AN | Confidentiality Integrity | H | P13, P22 |
| | | T23 | A | Integrity | H | P3 |
| Sensitive data | Confidentiality Integrity Accountability | T13 | A | Confidentiality Integrity | H | P13,P22 |
| | | T14 | A | Confidentiality Integrity | H | P13,P22 |
| | | T16 | AN | Confidentiality | M | P25 |
| | | T22 | A | Confidentiality | M | P3, P22 |
| | | T32 | A | Integrity | H | P3, P22 |
| | | T33 | A | Confidentiality | M | P26, P30 |
| | | T34 | A | Confidentiality | M | P26, P30 |
| | | T35 | N | Confidentiality Integrity | H | P25 |
| | | T45 | A | Accountability | M | P22, P24 |
| Sensitive process | Confidentiality Accountability | T16 | A | Confidentiality | M | P25 |
| | | T22 | A | Confidentiality | M | P3, P22 |
| | | T30 | A | Confidentiality | M | P3, P22 |
| | | T45 | A | Accountability | M | P1 |

For ease of comparing the mapping result, a detailed mapping list is given in Table 7-11.

Table 7-11 Detailed Mapping Result of Threat and Security Pattern

| Threat ID | Threat Name | Pattern ID | Pattern Name |
|---|---|---|---|
| T13 | Cross-Site Scripting (XSS) | P13, P22 | Intercepting Validator<br>Secure Base Action |
| T14 | SQL Injection | P13, P22 | Intercepting Validator<br>Secure Base Action |
| T16 | Network Eavesdropping | P25 | Secure Pipe |
| T22 | Disclosure of Confidential Data | P3, P22 | Authentication Enforcer<br>Secure Base Action |
| T23 | Data Tampering | P3,P22 | Authentication Enforcer<br>Secure Base Action |
| T33 | Session Hijacking | P26, P30 | Secure Session<br>Secure Facade |
| T34 | Session Replay | P26, P30 | Secure Session Secure Facade |
| T35 | Man in the Middle | P25 | Secure Pipe |
| T45 | User Denies Performing an Operation | P22, P24 | Secure Logger Secure Base Action |
| | | P1,P26 | Audit Interceptor Secure Facade |

### 7.5.2 Security Pattern Integration

Selected security patterns have been integrated into the system's high level architecture depicted as Figure 7-18.

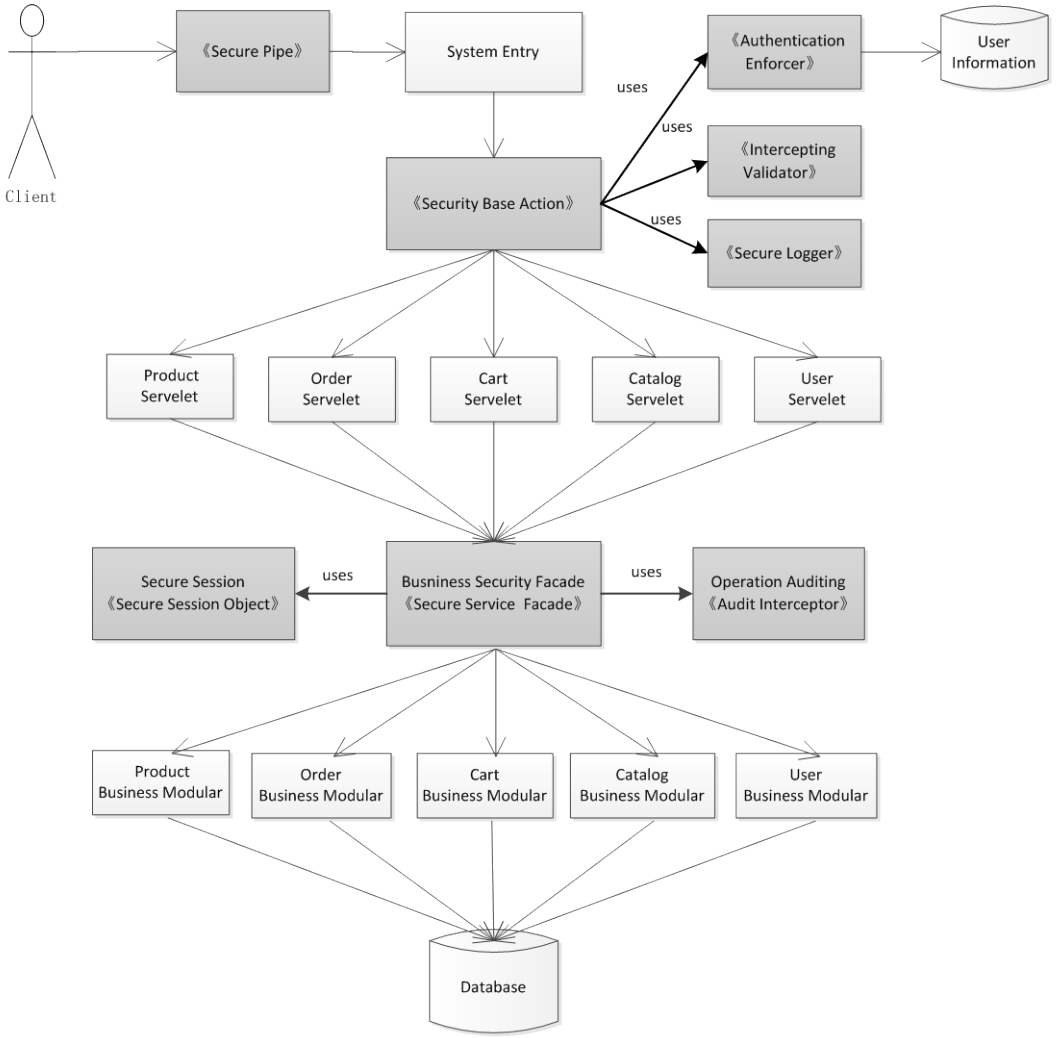



Figure 7-18 System High Level Architecture Integrated with Security Pattern

Security related components and methods are usually scattered across many or most of the system design artefacts which resulting in the reduced flexibility and reusability due to the inherent coupling between them. One of the efficient methods to solve this problem is to coordinate the security related components by providing a central access point for administrating security related functionalities. A Secure Base Action pattern is used as the single access point for this purpose. Figure 7-19 illustrates the Secure Base Action pattern as the access component to coordinate the security related components.

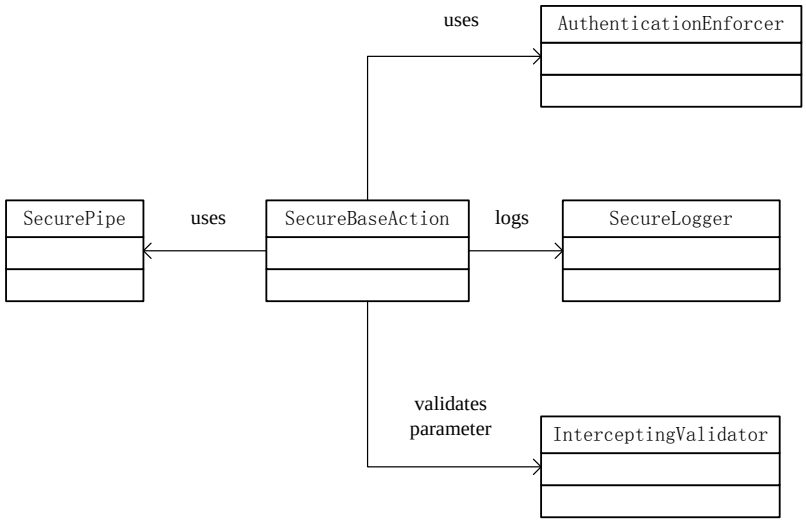



Figure 7-19 Class Diagram of Secure Base Action Pattern [158]

The Facade design of the original WebStoreApp can be upgraded into secure Facade by using Secure Service Facade as the access point in the business tier. Every access point has to authenticate and audit the operation with the help of combining Audit Interceptor and Secure Session patterns.

In this section, some of the security patterns are selected as the example to illustrate how they can be applied in the system with sample code.

### 7.5.2.1 Intercepting Validator Pattern

Several well-known attacks in web application, such as SQL injection, XSS and buffer overflow, are carried out by exploiting the vulnerabilities of lack of input validation.

Intercepting validator pattern [158] is one of the most important security patterns used in web-based application development since it serves as a filter to rule out all potential malicious code or malformed content from the input of users. These validations may include the validation on data type, data length, data formatting, data boundary and null-value handling.

Best practice indicates that validation checks have to be performed on the server side whether checking in client side is done or not. The reason lies in that it is easy to spoof and bypass the client side validation by using some proxy tools.

Suppose that the client needs to access the UserAccount resource, the system integrated

Intercepting Validator pattern is forced to validate the client's input according to the validation criteria provided by the pattern. Figure 7-20 and Figure 7-21 show the class diagram and sequence diagram when applying the Intercepting Validator pattern. The same can be applied to different resource access when there are input requests.

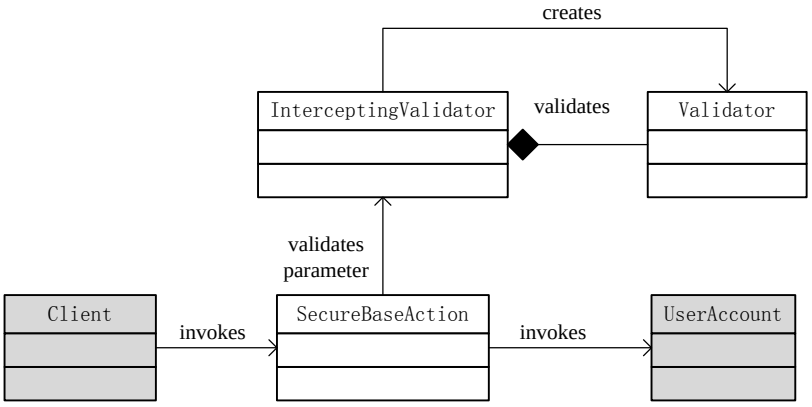



Figure 7-20 Class Diagram of Using Intercepting Validator Pattern to Validate the Request to UserAccount Class

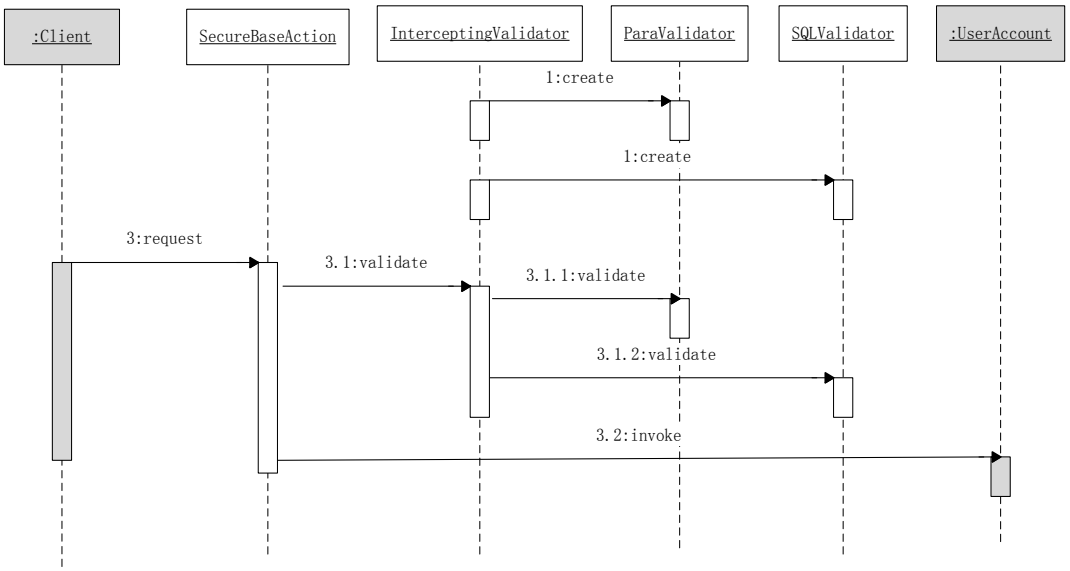



Figure 7-21 Sequence Diagram of Using Intercepting Validator Pattern to Validate the Request to UserAccount Class

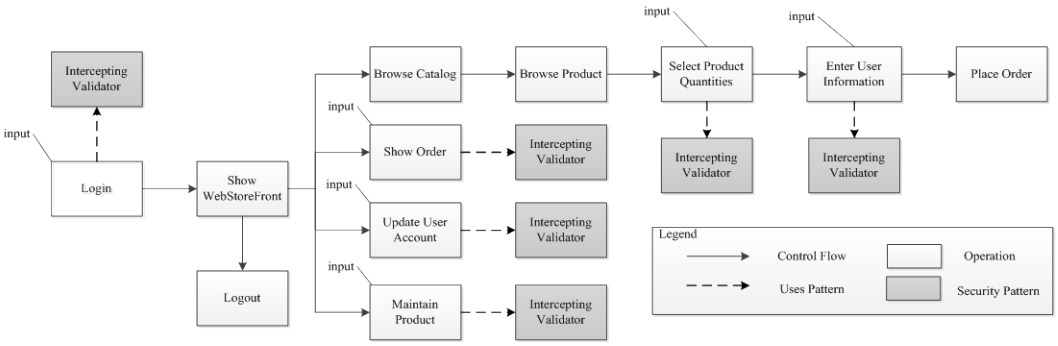



Figure 7-22 Block Diagram of the Main Component of WebStoreApp

A block diagram is shown in Figure 7-22 which describes the main component of the system where the Intercepting Validator patterns are used at the points where the user can input data, since these are the most crucial points in terms of security. List 7-1 shows the sample code of Intercepting Validator implementation with Apache Strut.

```
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
/**
* This code is based on Apache Struts examples.
* It requires a working knowledge of Struts, not explained here.
*/
public final class SecureBaseAction extends Action {
        public ActionErrors validate(ActionMapping actionmapping, HttpServletRequest request) {
                //perform basic input validation Validator validator = InterceptingValidator.getValidator
                  (actionmapping);
               ValidationErrors errors = validator.process(request);
               if(errors.hasErrors())
                        return InterceptingValidator. transformToActionErrors(errors);
               //For any additional externalised processing, use the
               //key 'additional_security_validator_identifier'
              //specified as 'parameter' attribute in action-mappings
```

```
            String externalisedProcessingKey = actionmapping.getParameter();
            ExternalisedValidator validatorEx = InterceptingValidator. getExternalisedValidator
            (externalisedProcessingKey);
            errors = validatorEx.process(request);
            if(errors.hasErrors())
                return InterceptingValidator.transformToActionErrors( errors);
            // Alternatively,
            // use 'additional_security_validator_identifier' to
            //specify a class that implements command pattern
            //and invoke the 'process' method on the instantiation
            try {
                Class cls = InterceptingValidatorUtil.loadClass(externalisedProcessingKey);
                Method method = InterceptingValidatrUtil.
                InterceptingValidator.invoke( cls, method, new Object[] {request}); }
            catch(Exception ex) {
                log("Invocation exception", ex);
                return InterceptingValidator.transformToActionErrors(ex); }
    }
}
```

List 7-1 Sample Code of SecureBaseAction class using InterceptingValidator with Apache Struts [158]

### 7.5.2.2 Secure Pipe

Sniffing, replay attack, man in the middle and network eavesdropping are common attacks occurred during the web based transactions. Data exposed when transmitting over the untrusted network is subject to be disclosed, modified or duplicated.

Secure Pipe pattern [158] plays a key role in protecting the confidentiality and integrity of data during the transmission against these attacks by establishing a secure communication between the client and the server. Secure Pipe pattern is composed of two components, client-side component and server-side component, which work together to build a secure communication. Generally, the components can be SSL

(Secure Socket Layer) or TSL (Transport Layer Security) libraries that the web browser of client side and application of server side use for secure communication.

Figure 7-23 and Figure 7-24 illustrate the structure of application scenario of Secure Pipe pattern while List 7-2 and List 7-3 show the sample code implementation using SSL in server side and client side respectively.

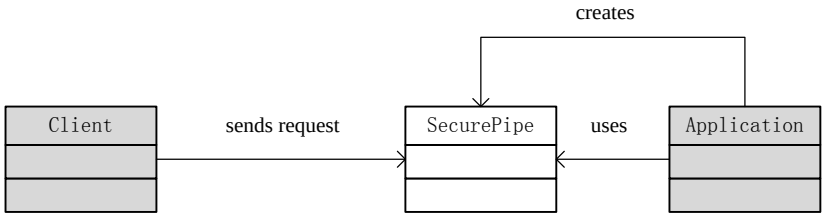



Figure 7-23 Class Diagram of Secure Pipe Pattern [158]

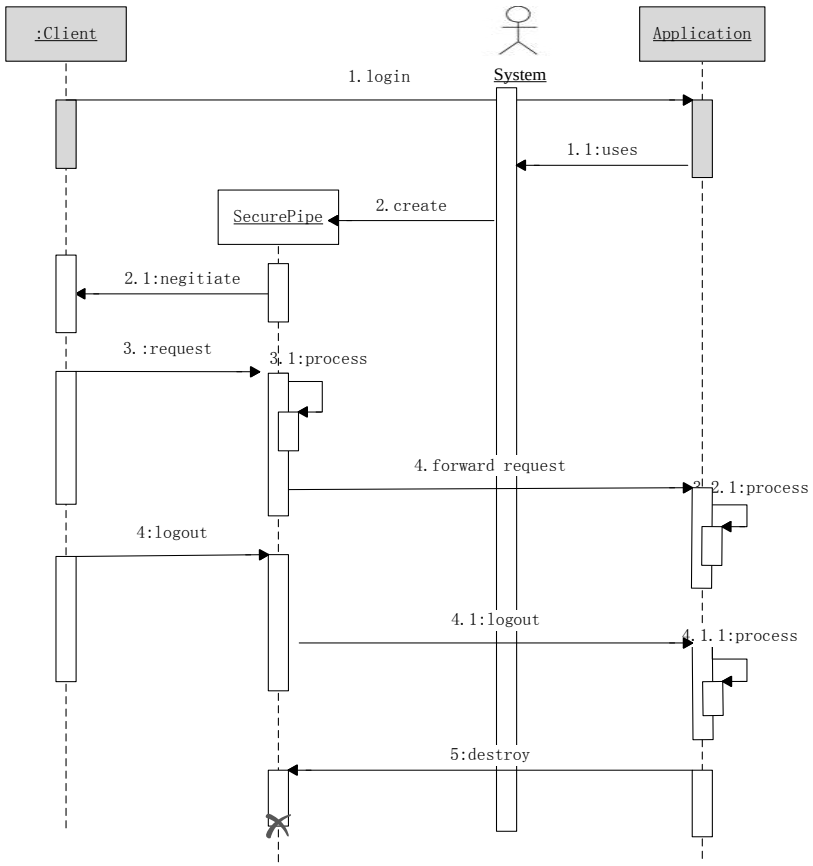



Figure 7-24 Sequence Diagram of Secure Pipe [158]

```
package com.csp.web.securepipe;

import java.io.*;

import java.net.*;

import java.rmi.server.*;

import java.security.KeyStore;

import javax.net.*;

import javax.net.ssl.*;

import com.sun.net.ssl.*;

import javax.security.cert.X509Certificate;

 /**

* This class creates RMI SSL connections.

*/

public class RMISSLServerSocketFactory

implements RMIServerSocketFactory, Serialisable {

        SSLServerSocketFactory ssf = null;

        /**

        * Constructor.

        */

        public RMISSLServerSocketFactory(char[] passphrase) {

        // set up key manager to do server authentication

        SSLContext ctx;

        KeyManagerFactory kmf;

        KeyStore ks;

        try {

                ctx = SSLContext.getInstance("SSL");

                // Retrieve an instance of an X509 Key manager

                kmf = KeyManagerFactory.getInstance("SunX509");
```

```
            //Get the keystore type.
            String keystoreType = System.getProperty( "javax.net.ssl.KeyStoreType");
            ks = KeyStore.getInstance(keystoreType);
            String keystoreFile = System.getProperty( "javax.net.ssl.trustStore");
           // Load the keystore.
            ks.load(new FileInputStream(keystoreFile), passphrase);
            kmf.init(ks, passphrase);
            passphrase = null;
            // Initialise the SSL context. ctx.init(kmf.getKeyManagers(), null, null);
            // Set the Server Socket Factory for getting SSL connections.
            ssf = ctx.getServerSocketFactory(); }
      catch(Exception e) {
            e.printStackTrace(); }
      }
     /**
     * Creates an SSL Server socketnad returns it.
     */
     public ServerSocket createServerSocket(int port)
     throws IOException {
              ServerSocket ss = ssf.createServerSocket(port);
     return ss; }
}
```

List 7-2 Creating a Secure RMI Server Socket Factory Using SSL [158]

```
Package com.csp.web.securepipe;
import java.io.*;
import java.net.*;
import java.rmi.server.*;
import javax.net.ssl.*;
```

```
public class RMISSLClientSocketFactory
implements RMIClientSocketFactory, Serializable {
    public Socket createSocket(String host, int port)
    throws IOException { SSLSocketFactory factory =
     (SSLSocketFactory)SSLSocketFactory.getDefault();
      SSLSocket socket = (SSLSocket)factory.createSocket(host, port) return socket; }
}
```

List 7-3 Creating a Secure RMI Client Socket Factory Using SSL [158]

#### 7.5.2.3 Authentication Enforcer

Poor authentication enables malicious users to access the system's resources for which they are unauthorised to do. It is dangerous that a user's credentials and related data to authenticate a web application are accessed by other users or co-existing applications. A web application must ensure that only valid users can access the restricted resources with proper authentication.

Authentication Enforcer pattern provides a centralised authentication enforcement by performing user authentication and encapsulating the implementation detail of the authentication mechanisms to ease evolving authentication requirements and facilitate reuse.

Figure 7-25 depicts the class diagram of Authentication Enforcer pattern. The responsibilities of each participant in Authentication Enforcer pattern are:

- Client. A client uses the Authentication Enforcer to authenticate a user.
- Authentication Enforcer. The Authentication Enforcer authenticates the user using the credentials passed in the Request Context.
- Request Context. The Request Context contains the user's credentials extracted from the protocol-specific request mechanism.
- Subject. The AuthenticationEnforcer creates a Subject instance that represents the authenticated user.

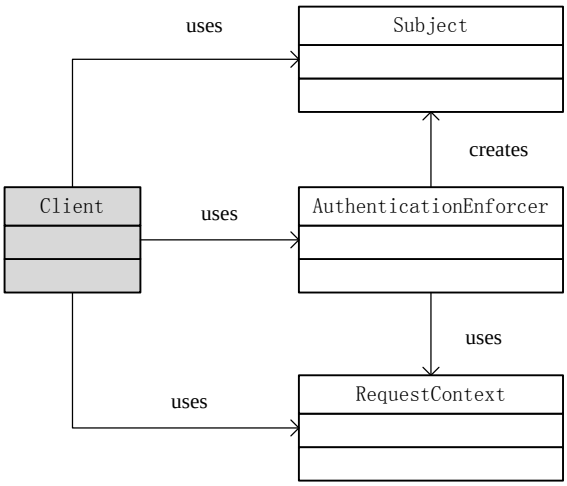



Figure 7-25 Class Diagram of Authentication Enforcer Pattern [158]

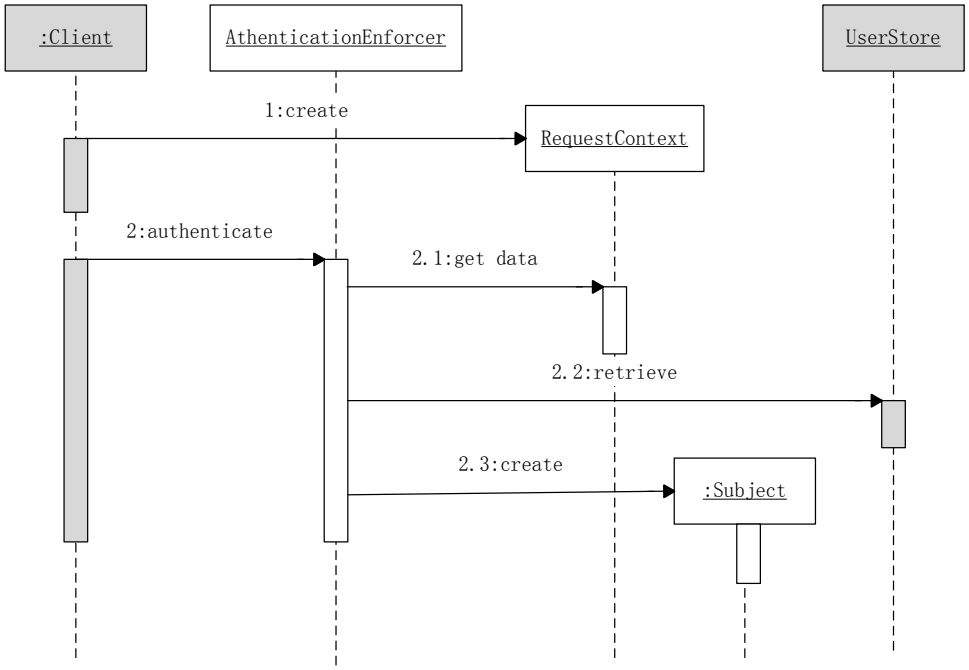



Figure 7-26 Sequence Diagram of Authentication Enforcer Pattern [158]

Figure 7-26 illustrates the sequence diagram in which the client is a Secure Base Action pattern that delegates to the Authentication Enforcer, which retrieves the appropriate user credentials from the UserStore. Upon successful authentication, the Authentication Enforcer creates a Subject instance for the requesting user and stores it in its cache.

```
package com.csp.web;
```

```
public class AuthenticationEnforcer {
    public Subject login(RequestContext request)
    throws InvalidLoginException {
            // 1. Instantiate the LoginContext
            // and load the LoginModule
            try {
                LoginContext ctx = new LoginContext("MyLoginModule",
                new WebCallbackHandler(request)); }
            catch(LoginException le) {
                System.err.println("LoginContext not created. "+ le.getMessage()); }
            catch(SecurityException se) {
                System.err.println("LoginContext not created. "+ se.getMessage()); }
            // 2. Invoke the Login method
              try {
                  ctx.login(); }
              catch(LoginException le) {
                  System.out.println("Authentication failed"); }
              System.out.println("Authentication succeeded");
              // Get the Subject
              Subject mySubject = ctx.getSubject();
              return mySubject;
        }
     }
```

List 7-4 Sample Code of Authentication Enforcer Pattern Using JAAS Authentication Strategy [158]

### 7.5.2.4 Final Integration Result

Security patterns are illustrated in the previous sections. In this section, the result of integration of the selected security patterns into the system model is shown in Figure 7-29. Figure 7-27 and Figure 7-28 specify the instantiation of security patterns in web

tier and business tier respectively. Based on the integration method proposed in Section 6.5.2, the class sets are integrated according the considerations.

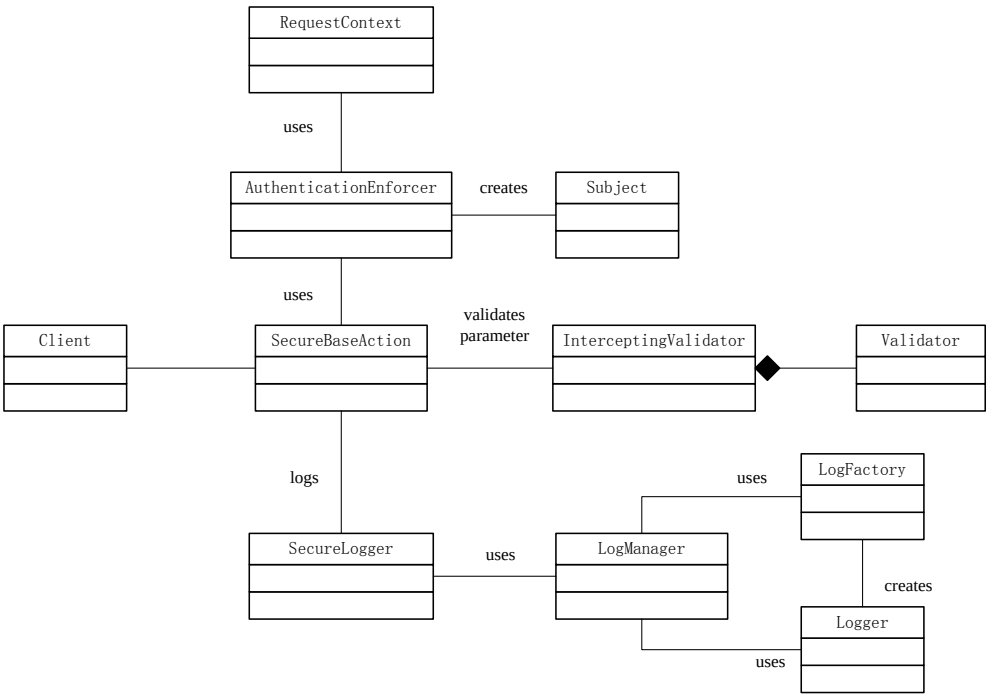



Figure 7-27 Class Diagram I of Secure Pattern Instantiation

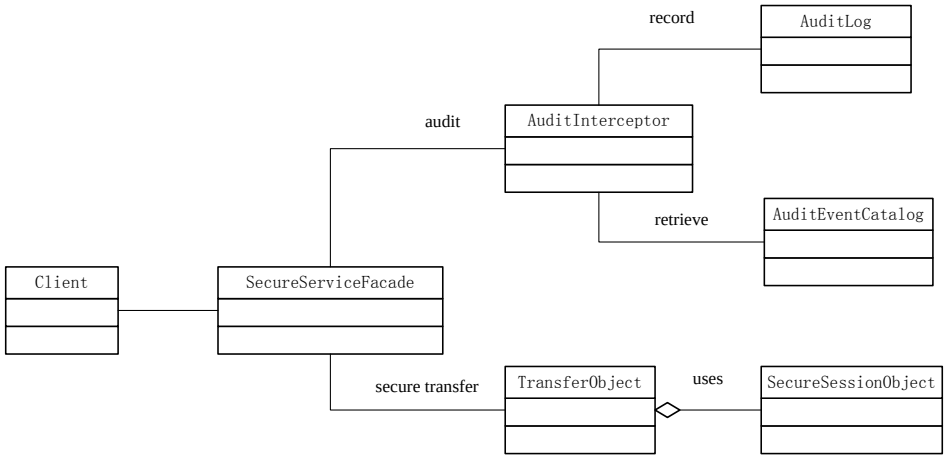



Figure 7-28 Class Diagram II of Secure Pattern Instantiation

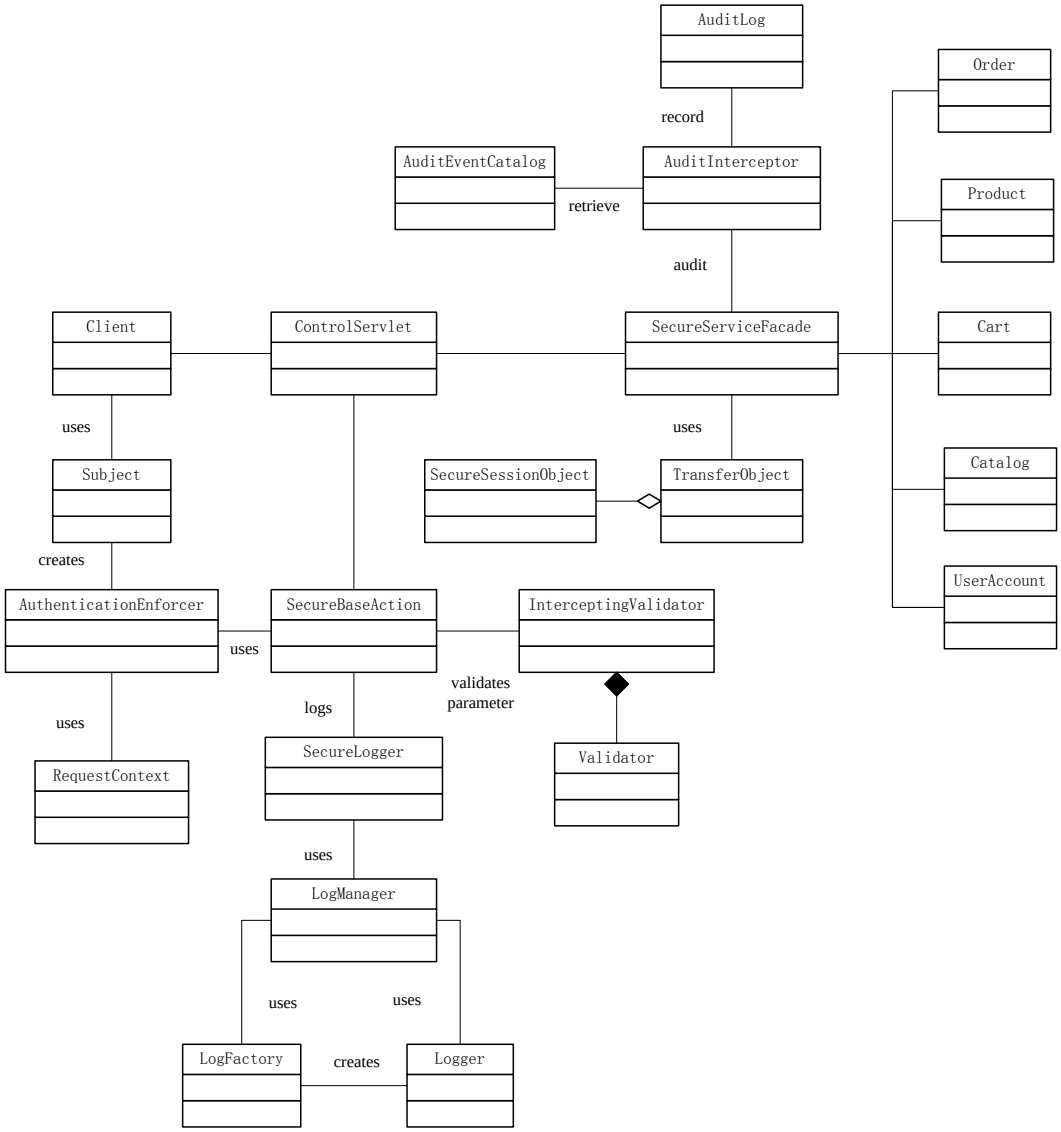



Figure 7-29 Class Diagram of System Model Integrated Security Patterns

From the final integration architecture, it can be concluded that security patterns can protect the system against the potential risks while increasing the system's complexity and may be take the efficiency as the cost. Therefore, it is important to make a trade-off decision between them for the system developer.

Security patterns have proven as the useful and efficient way to integrate security features into the system design. Even through there exist many versions of the application and across applications, these patterns will continue to grow and their implementation will be refined with the emerging of new security problems.

## 7.6 Discussion

The goal of this study is to provide a suitable, effective and reusable approach to enhancing security levels by reengineering web software. The previous sections of this chapter demonstrate how the proposed SEMDA approach is applied to the legacy web application WebStoreApp. This section discusses the proposed approach based on the following criteria:

- Effectiveness compared with other methods
- Efficiency on the needed user efforts
- Applicability for various context

### 7.6.1 Effectiveness Comparison with Other Methods

Security is a kind of quality which can only be defined in a relative way with respect to another system or by showing that a system satisfies some predefined security properties. The case study in this chapter demonstrates the latter by introducing security pattern to achieve the predefined security properties. The generated security enhanced system models indicates where the patterns should be integrated to protect which assets against which security problems. The experiment shows the proposed approach can lead to a security enhanced system on the model level.

The security improvement in this research highly depends on the security ability provided by security patterns. The ability to provide security by security pattern has been well evaluated and proved in some researches such as [26, 49, 69-71, 77, 185]. Take the approach in [71] as example, two e-commerce applications are designed in which one is developed using security patterns and other is built with the same requirement and techniques but without using security pattern. Security evaluation has been made to compare the quality of these two systems with the same category STRIDE attacks. The experiment results show that the non-secure application has a high risk of being affected by the attacks, whereas the secure application has a significantly lower risk. Therefore, this thesis does not conduct the research on what degree the security level has been leveraged for the given legacy system as such evaluation has been made in the mentioned approaches.

As described in Chapter 2, there are several approaches concerning security enhancement for existing systems. None of them concerns the security improvement for legacy system from the reengineering perspective. They focus either on some specific developing stage, such as architecture, design and implementation, or on some specific security aspects, e.g. authentication or authorisation. Even though some of them [29, 97, 153] provide systematic approach to evolve existing applications for security concerns both on requirement and architecture level, they assume the original design artefacts are at hand and address security problems directly upon them. However, it is not the case for most of the legacy systems. The proposed approach complements such information by providing a comprehensive solution involving a comprehension method to extract useful information especially security implementation information. Moreover, because of the encapsulation of security expertise solution in security pattern, the integration of security pattern into system design models makes more effective security guarantee.

Models representing higher level of abstraction play an important role in the proposed approach. Security pattern itself is reusable elements in the software design. Both make it possible to produce reusable and evolvable applications.

Although SEDMA approach involves several kinds of models and is based on model driven concepts, the “driven” is still in the preliminary stage. More general approach needs to be proposed to widen its range for utilising and the transformation rules are needed to facilitate the automation.

### 7.6.2 Efficiency on the Needed User Efforts

The proposed approach has been regarded as a semi-automatic process that involves a series of manual work and automatic transformation with toolset support. User efforts needed in the proposed approach is analysed from the activities involved in the whole evolution process.

In the process of legacy system understanding, user efforts are analysed as follows:

- Model extraction. This activity is achieved with the support of reverse engineering toolset. Appropriate tools can facilitate the UML model generation automatically. When it comes to object oriented applications, class diagram is

commonly supported in almost all of the reverse engineering tools. It is not he case for sequence diagram. Sequence diagram reverse engineered from source code using automatic tools can not represent the abstract object completely due to the specific implementation technique of programming language. For example, there may be intermediate result in the extracted diagram, while it should be hidden in the abstract level sequence diagram. Therefore, a manual revision to the extracted sequence diagram is required.

- Model dependency analysis. Different dependency among classes are analysed and defined which lays the foundation of subsequent model slicing. This process needs manual analysis. However, the proposed CSDG construction algorithm is likely to be implemented as a tool to generate the CSDG automatically.
- Model slicing. It is likely to realise the slicing automatically based on the proposed slicing algorithm with a given slicing criteria.
- Legacy system partition. Partition algorithm makes it possible to decompose the system into several clusters automatically.
- Security countermeasure detection. Identification of security implementation in the current design is a complicated process which involves a certain amount of manual intervention with the support of security expertise and experience.

In the process of security requirement elicitation, user efforts analysed as follows:

- Asset analysis. The proposed asset identification method is based on the semantic analysis of the system model with the security risk expertise and experience. Therefore, it needs manual intervention in this process.
- Threat analysis. Threat identification and quantification methods proposed in this research has been designed and implemented as a tool.
- Vulnerability analysis. This identification of vulnerability is implemented with an existing tool and quantified using CVSS.
- Risk assessment. With the output of asset, threat and vulnerability analysis, risk level is generated with the proposed formula automatically.

- Security evaluation. Manual work is needed to perform the evaluation together with the security expertise to determine whether the provided security satisfies the required security level.
- Security requirement elicitation. Security requirements can be produced automatically by formatting asset, threat, security features and risk level.

In the process of security integration and forward engineering, user efforts analysed as follows:

- Security pattern organisation. The proposed classification criteria make it possible to organise the security pattern automatically.
- Security pattern selection. It can be implemented by pattern search engine with the help of provided security ontology. OWL API as a Java API can be used to facilitate the process. With the provided selecting algorithm, it is likely to automate this process.
- Security pattern integration. Manual work is needed to match the pattern participants with the corresponding model elements.
- Code generation. Forward engineering toolset can facilitate this process automatically.

Security-driven software evolution is a complex and inherently knowledge intensive process which requires tons of domain knowledge involving software system knowledge, security engineering as well as expertise and experience from specialists. Even though some steps of the evolution process can be implemented to be automatic, manual intervention is still required to facilitate the understanding and analysis. The research in this direction is quite recent and far from producing a completely automatic transformation process.

### 7.6.3 Applicability for Various Context

The proposed security evolution approach is intended to be applied in web based applications coding by object oriented language. Even though several steps of the propose SEMDA approach have close relationships with object oriented techniques and

web applications, such as model extraction in Section 4.2 is automated by toolset which supports object oriented programming, security countermeasure detection in Section 4.6 concludes typical security mechanisms used in web applications, the proposed threat identification method in Section 5.2 is based on the environments where the web applications are hosted, the design ideas or activities in SEMDA can be used as a guideline for software practitioners to solve similar problems in other types of legacy systems.

## 7.7 Summary

The purpose of the legacy WebStoreApp case study is to illustrate that the proposed approach makes it possible to improve the security level during the software evolution. The process is achieved via the following steps:

- Understanding the legacy system through model extraction, model slicing and system partition techniques.
- Eliciting the security requirements via assessing the risk of legacy system by the analysing assets, threats and vulnerabilities.
- Improving the security level by integrating the appropriate security patterns into the legacy system.

# Chapter 8

# Conclusion and Future Work

### Objectives

- To summarise the whole thesis
- To revisit original contributions
- To evaluate the research with answers to the research questions
- To review the success criteria
- To illustrate the limitations of the work
- To outline future work

This chapter concludes the thesis by summarising the presented work and assessing to what extent the research aims and objectives set in Chapter 1 have been addressed. Firstly, a summary of the thesis is presented, then the significance of the contributions and the limitations issues are discussed, and finally directions for future work are suggested.

## 8.1 Summary of Thesis

The research on software security has become a hot topic in information domain. Many security methodologies and countermeasures have been developed and scattered all over the software development lifecycle to improve software security. However, most of the existing approaches to software security are limited to the integration of security features from the scratch of software development in the new designed software systems or maintenance of current used software by merely integrating some kind of security mechanisms. Systematic security improvement for legacy system is rarely

reported.

This thesis aims to integrate software security research with software evolution which is an attempt to improve the security level in legacy system using model driven approach. The proposed reengineering approach, SEMDA, involves many issues related to software reverse engineering for security, security requirement elicitation and security enhancement.

In this thesis, models representing the static and dynamic views are extracted from legacy code by toolset, which is the only possible system artefact. Once a series of models have been generated, they can serve as a foundation for the redevelopment of the legacy system to satisfy the new requirement. Security requirements can be elicited through analysing the risks to the legacy system and satisfied by integrating well-proven security pattern into system models. MDA forward engineering tools could facilitate the code generation from enhanced security models as well as provide round-trip reengineering for the legacy system. The case study confirms that the proposed SEMDA is a suitablel, effective and comprehensive approach.

## 8.2 Significance of Contributions and Evaluation

This thesis proposes solutions to integrate security engineering and software evolution, as observed in chapter 1. Specifically in Chapter 3, the thesis proposes a novel approach, SEMDA (Security-driven software Evolution using a Model Driven Approach). This section will revisit the original contributions described in Chapter 1:

**C1:** In Chapter 3, a novel software reengineering framework is created to integrate software security and software evolution for web application. This proposed framework shown in Figure 3-1 consists of legacy system extraction for security, security requirement elicitation and security enhancement with ontology-based security patterns.

**C2:** In Chapter 4, a method of legacy system understanding for security is proposed. A method of slicing models is presented on the basis of analysing different kinds of dependency relationships among classes and objects in UML diagrams. An intermediate representation diagram called CSDG is defined by combing the

advantages of class diagram and sequence diagram. The construction CSDG process is presented in List 4-1 and the model slicing algorithm based on the constructed CSDG is shown in List 4-2. An improved algorithm to decompose legacy system is developed based on the proposed CDSG and model slicing algorithm shown in List 4-3. A metric to measure the independence of a cluster is used and as a part of decomposition algorithm.

**C3:** In Chapter 5, a method of security requirement elicitation for web application is proposed. A classification method for web applications is presented via taking consideration of the environment where the application is hosted to ease the threat identification for software developer. The process is described in Section 5.2.2. An environment driven security requirement elicitation method for web application is proposed, based on which an approach to risk assessment using security vector for web application is proposed and its tool is developed to demonstrate the effectiveness of the proposed approach. The elicitation process is proposed in Section 5.2.

**C4:** In Chapter 6, a method of enhancing security level for web application is proposed. A multiple aspects method to organise security patterns is presented as security pattern is adopted as the security improvement methodology for security evolution. The detail of the method is described in 6.3.3. A security ontology for inter-relating elicited security requirements and security pattern is defined and realised in OWL in Section 6.4. Validation has been made on the proposed security ontology with a series of competence questions. The developed ontologies are shown in Figure 6-6 and matching algorithm is presented in List 6-8.

### 8.2.1 Research Questions Revisit

The evaluation of this study starts from answering the proposed research questions presented in Chapter 1:

**How can security patterns be used to meet the security requirements elicited by risk analysis through model driven approach to evolve the security for web software applications?**

The question has been answered in general by proposing an approach called SEMDA. A collection of research questions are defined to refine the whole question in detail.

*RQ1: Why is there a need for a security driven software evolution approach?*

Legacy systems are threatened by a lot of security problems due to their old design and lack of security consideration. (Section 1.1)

*RQ2: How models are used to direct the whole process?*

Models and modelling techniques as the main artefacts and activities are used at every stages.

- *How may the models be extracted from source code in legacy systems?*

  Automatic tools are used to extract the initial diagrams, and then manual intervention is needed to correct and revise them into a form that can be used to the next stage. (Section 4.2.3)

- *What type of models is required to reengineer the legacy system?*

  Static diagram representing the system structure and dynamic diagram representing the system behaviour are chosen as the models in the proposed approach, which are class diagram and sequence diagram in UML. (Section 4.2.2)

- *How may the models be used to reengineer legacy system?*

  System design models are extracted by reverse engineering techniques. Extracted models are used to construct intermediate graph CSDG (Section 4.3), which is used to facilitate the system understanding (Section 4.4). System domain models are achieved by analysing risks (Section 5.2). Security enhanced design models are finally generated by combining the system design model, security requirements domain model and security pattern model (Section 6.5).

*RQ3: How may security requirements be elicited from the legacy systems?*

Underlying requirements may be extracted by applying risk analysis for legacy system. (Section 6.4)

- *How may the risk be assessed for legacy system?*

A security vector based method is proposed to assess the risk level for the legacy system taking asset, vulnerability and threat into consideration. (Section 5.1)

- *What kind of information is needed to represent the security requirement?*

  After risk analysis, a list with columns of asset, threat and risk level is generated which is treated as the security requirement format in this thesis. (Section 5.4)

*RQ4: How may the elicited security requirements be satisfied by security patterns?*

Security patterns have been proven as a useful and effective way to improve security level in the system design. (Section 6.3)

- *How may the "right" security patterns be found for the elicited security requirement?*

  Section 6.4 shows an ontology is developed to map the security pattern with the security requirements.

- *How may the security patterns be integrated into legacy system model?*

  Section 6.5 describes the integration process of the selected security patterns diagrams with the system diagrams.

*RQ5: How can the proposed approach be validated?*

Chapter 7 shows how the SEMDA approach is applied in a typical case study.

### 8.2.2 Success Criteria Revisit

In Chapter 1, a set of criteria are proposed to judge the success of the proposed approach in this research. In this section, detailed analysis of the proposed approach is presented based on these criteria.

- *What type of legacy systems is the proposed approach applicable?*

  As description in Section 7.6.3, the SEMDA approach is suitable for the web based legacy systems whose source codes are available. Case study shows that SEMDA approach can extract models from the source code, analyse and apply the proposed techniques to the models to achieve the final goal.

- *Are the extracted models consistent to the original design and easy to*

*understand?*

The answer is positive. Even though extracting models from source code may result in information lost, the remained is the structural information preserving the necessary structure and traceability which lay the foundation for the new design. The extracted models are easy to understand by using UML as well. The conducted case study shows the positive evidence to this conclusion.

- *Is the security requirement elicitation framework able to reflect system's detailed level security requirement under the current environment?*

  The answer is yes. Detailed security requirement is specified using 4-tuple *<Asset, Threat, Security Attribute, Priority>* which means that which asset is protected in the security requirement against the threatening from which threat(s) by violating which security attribute and what kind of risk level.

- *How can the proposed method be able to address the elicited security problems and provide implementation issues supporting for development?*

  Security pattern has been proven to be a useful and efficient way to solve the occurred security problem. Ontology, as a tool of knowledge representation and inferring, provides security pattern semantic ability. Moreover, due to the OWL representation, the security patterns are available in a machine readable format and it is expected to be utilised in the system automatically.

- *Is the security enhanced models reliable to perform forward engineering?*

  The answer is positive. Lots of forward engineering tools have been developed and introduced in Section 6.6. As long as the security patterns have been refined and documented with enough information, it is possible to generate the code from the security enhanced models automatically. The case study shows the instantiation of the security pattern.

- *Is the proposed approach feasible to realise? For example, it is possible to design and implement a real tool to demonstrate the approach.*

  The answer is yes. During every phase of the proposed approach, various algorithms have been developed and specified. It is possible to integrate the

implementation of each algorithm and provide a unified interface for real practice.

## 8.3 Limitations

Besides the success criteria mentioned above, it is believed that the proposed approach had additional successes as well as some challenges.

- *Dynamic diagrams cannot always be extracted from source code completely.*

  Some models cannot satisfactorily be extracted from source code. These models are highly dependent on user interaction to identify external actors and their roles or are behaviour models of highly reactive systems with many external events. A developer/user community must be available in order to identify these external actors and the roles that they play in regards to the system or these actors with their roles must be available in up-to-date documentation.

- *Security mechanisms may demand manual work and become time consuming*

  During the detection stage, a great number of security criteria will be discovered and determined. Due to the complicated implementation of security, security expertise is needed. If mistakes exist in this process, it will affect the decision making.

- *The poor quality of the legacy system may affect the efficiency of the proposed approach.*

  In this research, the legacy system is analysed by using reverse engineering tools. If the legacy systems are designed by standard programming, such as comments, document and coding format, etc., the proposed approach will be more effective. Otherwise, human intervention is needed to improve the quality of model extraction.

- *The category of security pattern has not reached the greatest extent and needs to be enriched.*

  Security pattern based design will be limited when there doesn't exist the

applicable security patterns to a specific security problem. The application of classic security design principles is a good alternative in such circumstance. However, with the more and more security knowledge being encapsulated into security patterns, it is certainly that security problems can be solved easily with the help of security patterns. The pattern based approach used in this thesis benefit from the rich security patterns. More importantly than all of that, it provides the idea of security knowledge reuse.

## 8.4 Future Work

In terms of the discussion with respect to the research questions, the research hypothesis, the original contributions, the criteria of success, and limitations, it can be concluded that SEMDA approach is a novel and systematic methodology for reengineering software for security improvement. The case study in Chapter 7 demonstrates the overall security reengineering process. Nevertheless, the research work in this thesis has not reached its endpoint. A series of future work can be enhanced and extended based on the current study.

- Based on the presented model slicing approach in Section 4.4, sequence diagram can be further refined to several execution scenarios which can be integrated into the construction of CSDG for illustrating the system precisely.
- The proposed threat elicitation approach is based on the web application classification which can be extended to other type of applications, for example traditional desktop application, embedded application etc.
- The proposed security ontology, designed focusing on the mapping of security requirements and security patterns in the current work, can be instantiated using OWL API (e.g. Jena, a java plugin to process OWL ontology) to implement the processing of OWL ontology with graphical interface.
- Security pattern integration relies on the precise documentation of security pattern repository. There are 32 security patterns in the pattern repository in the current work which can be extend with more and precise security patterns

in the future study.

- In order to better evaluate the effectiveness of the proposed approach, various case studies should be carried out. Especially, development an integrated toolset to ease the use of the proposed method could be another research project in the future works.
- Due to the crosscutting nature of AOP, it is a good choice to deal with security evolution only if security is treated as one of the system's concerns. Modelling security concerns as security aspects makes it possible to separate the security concerns with other system concerns which will generate a reusable and evolvable secure system. Therefore, implement security with AOP could be a meaningful research topic in the further study.

# References

[1] M. Alam, R. Breu and M. Hafner, "Modeling Permissions in a (U/X)Ml World," in *The First International Conference on Availability, Reliability and Security (ARES'06)*, pp. 8 2006.

[2] M. M. Alam, R. Breu and M. Breu, "Model Driven Security for Web Services (MDS4WS)," in *Proceedings of 8th International Multitopic Conference (INMIC'04)*, pp. 498-505, 2004.

[3] C. J. Alberts, A. J. Dorofee, J. F. Stevens and C. Woody, *Introduction to the OCTAVE Approach*, Software Engineering Institute, Available: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=51546, 2003.

[4] I. Alexander, "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software*, vol. 20, no. 1, pp. 58-66, 2003.

[5] E. G. Amoroso, *Fundamentals of Computer Security Technology*: Prentice-Hall, Inc., 1994.

[6] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*: Wiley, 2001.

[7] ArgoUML, *ArgoUML-Overview*, Available: http://argouml.tigris.org/, 2008.

[8] D. Atkins, *Internet Security Professional Reference*: New Riders Publishing, 1997.

[9] C. Bachman, "A Case for Reverse Engineering," *Datamation*, vol. 34, no. 13, pp. 49-64, 1988.

[10] C. Banerjee and S. Pandey, "Software Security Rules, SDLC Perspective," *International Journal of Computer Science and Information Security*, vol. 6, no. 1, pp. 123-128, 2009.

[11] O. Barais, A. F. Le Meur, L. Duchien and J. Lawall, "Software Architecture Evolution in Software Evolution," ed: Springer, pp. 233-262, 2008.

[12] D. Basin, J. Doser and T. Lodderstedt, "Model Driven Security for Process-Oriented Systems," in *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, Como, Italy, pp. 100-109, 2003.

[13] C. Bennett, D. Myers, M. A. Storey, D. M. German, D. Ouellet, M. Salois, *et al.*, "A Survey and Evaluation of Tool Features for Understanding Reverse‐Engineered Sequence Diagrams," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 291-315, 2008.

[14] K. H. Bennett and V. T. Rajlich, "Software Maintenance and Evolution: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, New Yok, USA, pp. 73-87, 2000.

[15] B. Best, J. Jurjens and B. Nuseibeh, "Model-based Security Engineering of Distributed Information Systems using Umlsec," in *29th International Conference on Software Engineering(ICSE'07)*, Minneapolis, pp. 581-590, 2007.

[16] B. Blakley and C. Heath, "Security Design Patterns Technical Guide - Version 1," the Open Group, 2004.

[17] A. Braga, C. Rubira and R. Dahab, "Tropyc: A Pattern Language for Cryptographic Software," in *Proceedings of the fifth conference on pattern languages of programming (PLoP ’98)*, 1998.

[18] M. L. Brodie and M. Stonebraker, *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*: Morgan Kaufmann Publishers Inc., 1995.

[19] BSI, BS7799 - Code of Practice for Information Security Management, British Standards Institute, 1999.

[20] A. Buecker, P. Ashley, M. Borrett, M.Lu, S. Muppidi and N. Readshaw, *Understanding SOA Security Design and Implementation*, IBM Redbooks, 2007.

[21] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal and P. Sommerlad, *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*: John wiley & sons, 1996.

[22] K. Buyens, B. De Win and W. Joosen, "Resolving Least Privilege Violations in

Software Architectures," in *ICSE Workshop on Software Engineering for Secure Systems(SESS'09)*, Vancouver, pp. 9-16, 2009.

[23] CERT, *CERT Statistics*, Computer Emergency Readiness Team Coordination Centre, Available: http://www.cert.org/stats/cert_stats.html, 2008.

[24] CERT, *(CERT/CC) 2004 E-Crime Watch*, Computer Emergency Readiness Team Coordination Centre, Available: http://www.cert.org/about/ecrime.html 2004.

[25] F. Chen, "Model driven Software Modernisation," Ph.D Thesis, De Montfort University, 2007.

[26] B. H. Cheng, S. Konrad, L. A. Campbell and R. Wassermann, "Using Security Patterns to Model and Analyse Security Requirements," *IEEE Workshop on Requirements for High Assurance Systems*, pp. 13-22, 2003.

[27] E. J. Chikofsky and J. H. C. II, "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13-17, 1990.

[28] D. Cock, K. Wouters, D. Schellekens, D. Singelee and B. Preneel, "Threat Modelling for Security Tokens in Web Applications," in *Communications and Multimedia Security*, ed: Springer US,vol. 175, pp. 183-193, 2005.

[29] D. Cotroneo, A. Mazzeo, L. Romano and S. Russo, "An Architecture for Security-Oriented Perfective Maintenance of Legacy Software," *Information and Software Technology*, vol. 45, no. 9, pp. 619-631, 2003.

[30] DCID, "Protecting Sensitive Compartmented Information within Information Systems," Director of Central Intelligence Directive 6/3, Available: http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm, Mar. 2011.

[31] R. De Landtsheer and A. Van Lamsweerde, "Reasoning about Confidentiality at Requirements Engineering Time," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM Sigsoft International Symposium on Foundations of Software Engineering*, pp. 41-49, 2005.

[32] B. De Win, R. Scandariato, K. Buyens, J. Grégoire and W. Joosen, "On the

Secure Software Development Process: Clasp, SDL and Touchpoints Compared," *Information and Software Technology*, vol. 51, no. 7, pp. 1152-1171, 2009.

[33] P. T. Devanbu and S. Stubblebine, "Software Engineering for Security: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, pp. 227-239, 2000.

[34] A. Dikanski, C. Emig and S. Abeck, "Integration of a Security Product in Service-Oriented Architecture," in *Third International Conference on Emerging Security Information, Systems and Technologies*, Greece, pp. 1-7, 2009.

[35] G. Dobson and P. Sawyer, "Revisiting Ontology-based Requirements Engineering in the Age of the Semantic Web," in *Proceedings of the International Seminar on Dependable Requirements Engineering of Computerised Systems at NPPs*, Halden, 2006.

[36] M. Donner, "Toward a Security Ontology," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 6-7, 2003.

[37] L. M. Duarte, J. Kramer and S. Uchitel, "Model Extraction using Context Information," in *Proceedings of 9th International Conference on Model Driven Engineering Languages and Systems*, pp. 380-394, 2006.

[38] P. El Khoury, A. Mokhtari, E. Coquery and M. S. Hacid, "An Ontological Interface for Software Developers to Select Security Patterns," in *19th International Workshop on Database and Expert Systems Application(DEXA '08)*, pp. 297-301, 2008.

[39] H. F. El Yamany, M. A. Capretz and D. S. Allison, "Intelligent Security and Access Control Framework for Service-Oriented Architecture," *Information and Software Technology*, vol. 52, no. 2, pp. 220-236, 2010.

[40] R. J. Ellison, "Attack Trees," SET Carnegie Mellon University, Available: http://web.cs.du.edu/~ramki/papers/attackGraphs/EllisonAttackTrees.pdf, 2005.

[41] C. Emig, F. Brandt, S. Abeck, J. Biermann and H. Klarl, "An Access Control Metamodel for Web Service-Oriented Architecture," in *International*

*Conference on Software Engineering Advances(ICSEA'07)*, pp. 57-57, 2007.

[42] C. Emig, F. Brandt, S. Kreuzer and S. Abeck, "Identity as a Service–Towards a Service-Oriented Identity Management Architecture," in *Dependable and Adaptable Networks and Services*, ed: Springer, pp. 1-8, 2007.

[43] J.-M. Favre, "Foundations of Model (Driven)(Reverse) Engineering: Models," in *Dagstuhl Seminar Proceedings of Seminar on Language Engineering for Model-Driven*, Dagsthul, Germany, pp. 1-31, 2004.

[44] S. Fenz and A. Ekelhart, "Formalizing Information Security Knowledge," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pp. 183-194, 2009.

[45] E. Fernandez-Buglioni, *Security Patterns in Practice: Designing Secure Architectures using Software Patterns*: John Wiley & Sons, 2013.

[46] E. B. Fernandez, "Metadata and Authorisation Patterns," Available: http://www.cse.fau.edu/wed/MetadataPatterns.pdf, 1996.

[47] E. B. Fernandez, M. M. Larrondo-Petrie, N. Seliya, N. Delessy-Gassant and M. Schumacher, "A Pattern Language for Firewalls," in *Proceedings of Pattern Languages of Program(PLoP'03)*, 2003.

[48] E. B. Fernandez and J. Sinibaldi, "More Patterns for Operating System Access Control," in *Proceedings of Euro Pattern Languages of Program(PLoP'03)*, pp. 381-398, 2003.

[49] E. B. Fernandez, N. Yoshioka, H. Washizaki and M. VanHilst, "Measuring the Level of Security Introduced by Security Patterns," in *2010 International Conference on Availability, Reliability and Security*, Krakow, pp. 565-568, 2010.

[50] D. Firesmith, "Analysing the security significance of system requirements," in *Symposium on Requirements Engineering for Information Security (SREIS'05), in Conjunction with RE 05-13th IEEE International Requirements Engineering Conference*, Paris, France, 2005.

[51] W. Ford and M. S. Baum, *Secure Electronic Commerce*: Prentice-Hall

Incorporation, Upper Saddle River, 1997.

[52] M. Fowler, *Uml Distilled: A Brief Guide to the Standard Object Modeling Language*: Addison-Wesley Professional, 2004.

[53] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Pearson Education, 1994.

[54] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*: Addison-Wesley, 1995.

[55] M. Gasmi, M. Derdour and N. G. Zine, "Towards a Security Meta-model for Software Architectures," *The International Arab Journal of Information Technology,* 2011.

[56] G. Georg, I. Ray, K. Anastasakis, B. Bordbar, M. Toahchoodee and S. H. Houmb, "An Aspect-Oriented Methodology for Designing Secure Applications," *Information and Software Technology,* vol. 51, no. 5, pp. 846-864, 2009.

[57] D. P. Gilliam, T. L. Wolfe, J. S. Sherif and M. Bishop, "Software Security Checklist for the Software Life Cycle," in *Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Conference(WET ICE '03)*, pp. 243-248, 2003.

[58] A. Gomez-Perez, M. Fernández-López and O. Corcho, *Ontological Engineering*: Springer Heidelberg, 2004.

[59] D. Gries and F. B. Schneider, *Monographs in Computer Science*: Springer, 1993.

[60] D. Gross and E. Yu, "From Non-Functional Requirements to Design through Patterns," *Requirements Engineering,* vol. 6, no. 1, pp. 18-36, 2001.

[61] T. R. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing?," *International Journal of Human-Computer Studies,* vol. 43, no. 5, pp. 907-928, 1995.

[62] H. Guan, W. Chen, L. Liu and H. Yang, "Estimating Security Risk for Web Applications using Security Vectors," *Journal of Computers,* vol. 23, no. 1, pp. 54-69, 2012.

[63] H. Guan, W. Chen, L. Liu and H. Yang, "Environment-Driven Threats Elicitation for Web Applications," in *Agent and Multi-Agent Systems: Technologies and Applications*, ed: Springer, pp. 291-300, 2011.

[64] H. Guan, W. R. Chen, H. Li and J. Wang, "STRIDE–Based Risk Assessment for Web Application," *Applied Mechanics and Materials*, vol. 58, pp. 1323-1328, 2011.

[65] M. Hafiz, P. Adamczyk and R. E. Johnson, "Organising Security Patterns," *IEEE Software*, vol. 24, no. 4, pp. 52-60, 2007.

[66] M. Hafiz and R. E. Johnson, "Security Patterns and Their Classification Schemes," University of Illinois at Urbana-Champaign Department of Computer Science, 2006.

[67] M. Hafiz and R. E. Johnson, "Evolution of the Mta Architecture: the Impact of Security," *Software Practice and Experience*, vol. 38, no. 15, pp. 1569-1599, 2008.

[68] M. Hafner, M. Memon and R. Breu, "SeAAS-A Reference Architecture for Security Services in SOA," *Journal of Universal Computer Science*, vol. 15, no. 15, pp. 2916-2936, 2009.

[69] S. Halkidis, A. Chatzigeorgiou and G. Stephanides, "A Qualitative Evaluation of Security Patterns," in *Information and Communications Security*, ed: Springer Berlin Heidelberg,vol. 3269, pp. 132-144, 2004.

[70] S. T. Halkidis and E. Chatzigeorgiou, "A Practical Evaluation of Security Patterns," in *Proceedings of the International Conference on Artificial Intelligence and Digital Communications*, pp. 1-8, 2006.

[71] S. T. Halkidis, N. Tsantalis, A. Chatzigeorgiou and G. Stephanides, "Architectural Risk Analysis of Software Systems Based on Security Patterns," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 129-142, 2008.

[72] A. Hamou-Lhadj and T. C. Lethbridge, "A Survey of Trace Exploration Tools and Techniques," in *Proceedings of the 2004 Conference of the Centre for*

*Advanced Studies on Collaborative Research*, Markham, Ontario, Canada, pp. 42-55, 2004.

[73] J. Han, R. Kowalczyk and K. M. Khan, "Security-Oriented Service Composition and Evolution," in *13th Asia Pacific Software Engineering Conference (APSEC'06)*, pp. 71-78, 2006.

[74] P. Harmon, The OMG's Model Driven Architecture and BPM. *Newsletter of Business Process Trends*, vol. 2, no. 5, pp. 1-11, 2004.

[75] D. Hatebur, M. Heisel and H. Schmidt, "Analysis and Component-based Realization of Security Requirements," in *Third International Conference on Availability, Reliability and Security (ARES'08)*, pp. 195-203, 2008.

[76] A. Herzog, N. Shahmehri and C. Duma, "An Ontology of Information Security," *International Journal of Information Security and Privacy (IJISP)*, vol. 1, no. 4, pp. 1-23, 2007.

[77] T. Heyman, K. Yskout, R. Scandariato and W. Joosen, "An Analysis of the Security Patterns Landscape," in *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, pp. 3, 2007.

[78] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies," *Journal of Semantic Web*, vol. 2, no. 1, pp. 11-21, 2011.

[79] M. Howard and S. Lipner, *The Security Development Lifecycle (SDL): A Process for Developing Demonstrably More Secure Software*: Microsoft Press, 2006.

[80] ISO/IEC, Code of Practice for Information Security Management, ISO/IEC 17799-272002, 2005.

[81] IST, "An Introduction to Computer Security—the NIST Handlebook," ed: TechnicalReport: NIST (National Institute of Standards and Technology), Special Publication 800-12, 1995.

[82] L. Jiang, H. Chen and F. Deng, "A Security Evaluation Method Based on STRIDE Model for Web Service," in *2nd International Workshop on Intelligent Systems and Applications (ISA'10)*, Wuhan, China, pp. 1-5, 2010.

[83] J. Jurjens, "UMLsec: Extending UML for Secure Systems Development," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, pp. 412-425, 2002.

[84] J. Jurjens, "Developing High-Assurance Secure Systems with Uml: A Smartcard-based Purchase Protocol," in *Proceedings of Eighth IEEE International Symposium on High Assurance Systems Engineering*, pp. 231-240, 2004.

[85] J. Jürjens, "Towards Development of Secure Systems Using UMLsec," in *Fundamental Approaches to Software Engineering*, ed: Springer Berlin Heidelberg,vol. 2029, pp. 187-200, 2001.

[86] J. Jurjens, J. Schreck and P. Bartmann, "Model-based Security Analysis for Mobile Communications," in *Proceedings of the 30th International Conference on Software Engineering*, Leipzig, Germany, pp. 683-692, 2008.

[87] K. M. Khan and J. Han, "Composing Security-aware Software," *IEEE Software*, vol. 19, no. 1, pp. 34-41, 2002.

[88] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, *et al.*, "Aspect-Oriented Programming," in *11th European Conference on ECOOP'97 — Object-Oriented Programming*, pp. 220-242, 1997.

[89] D. M. Kienzle and M. C. Elder, "Security Patterns for Web Application Development," Final Technical Report: University of Virginia, 2002.

[90] D. M. Kienzle, M. C. Elder, D. Tyree and J. Edwards-Hewitt, "Security patterns repository version 1.0," *DARPA, Washington DC*, 2002.

[91] D. M. Kienzle, M. C. Elder, D. Tyree and J. Edwards-Hewitt, "Security Patterns Repository Version 1.0," Available: http://www.scrypt.net/~celer/securitypatterns/repository.pdf, 2002.

[92] M. Kis, "Information Security Antipatterns in Software Requirements Engineering," in *9th Conference of Pattern Languages of Programs (PloP)*, 2002.

[93] A. G. Kleppe, J. B. Warmer and W. Bast, *Mda Explained: The Model Driven*

*Architecture: Practice and Promise*: Addison-Wesley Professional, 2003.

[94] P. B. Kruchten, "The 4+ 1 View Model of Architecture," *IEEE Software,* vol. 12, no. 6, pp. 42-50, 1995.

[95] J. T. Lallchandani and R. Mall, "Slicing UML Architectural Models," *ACM SIGSOFT Software Engineering Notes,* vol. 33, no. 3, pp. 4, 2008.

[96] J. T. Lallchandani and R. Mall, "A Dynamic Slicing Technique for Uml Architectural Models," *IEEE Transactions on Software Engineering,* vol. 37, no. 6, pp. 737-771, 2011.

[97] R. C. Laney, J. Van der Linden and P. Thomas, "Evolution of Aspects for Legacy System Security Concerns," in *Proceedings of International Conference on Aspect-Oriented Software Development (AOSD'04)*, pp. 35, 2004.

[98] J. Lasheras, R. Valencia-García, J. T. Fernández-Breis and A. Toval, "Modelling Reusable Security Requirements based on an Ontology Framework," *Journal of Research & Practice in Information Technology,* vol. 41, no. 2, pp. 119-133, 2009.

[99] M. A. Laverdiere, A. Mourad, A. Hanna and M. Debbabi, "Security Design Patterns: Survey and Evaluation," in *Conference on Electrical and Computer Engineering (CCECE '06)*, Ottawa, pp. 1605-1608, 2006.

[100] F. Lee Brown, J. Di Vietri, G. Diaz de Villegas and E. Fernandez, "The Authenticator Pattern," in *Conference on Pattern Languages of Programming (PLoP'99)*, 1999.

[101] S. Lehtonen and J. Pärssinen, "A Pattern Language for Key Management," in *Proceedings of Pattern of Language Porgramming(PLoP'01)*, 2001.

[102] L. Liu, E. Yu and J. Mylopoulos, "Security and Privacy Requirements Analysis within a Social Setting," in *IEEE Joint International Conference on Requirements Engineering*, California, USA, pp. 151-161, 2003.

[103] L. Liu, E. Yu and J. Mylopoulos, "Analysing Security Requirements as Relationships Among Strategic Actors," in *Symposium on Requirements Engineering for Information Security (SREIS'02)*, Raleigh, North Carolina,

2002.

[104] T. Lodderstedt, D. A. Basin and R. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," in *Proceedings of the 5th International Conference on The Unified Modeling Language*, Germany, pp. 426-441, 2002.

[105] J. Luo, L.Zhang and J. Sun, "A Hierarchical Decomposition Method for Object-Oriented Systems Based on Identifying Omnipresent Clusters," in *Proceedings of the 21st IEEE International Conference on Software Maintenance(ICSM'05)*, Budapest, Hungary, pp. 647-650, 2005.

[106] MAGERIT, *version 2 - Methodology for Information Systems Risk Analysis and Management - Book I - The Method*, Available: http://www.seap.minhap.gob.es/dms/es/publicaciones/centro_de_publicaciones_de_la_sgt/Monografias0/parrafo/01111111111111111118/text_es_files/Magerit-v2-book-I.pdf, 2006.

[107] Q. Mahmoud, "Security Policy: A Design Pattern for Mobile Java Code," in *Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP'00)*, 2000.

[108] P. K. Manadhata and J. M. Wing, "An Attack Surface Metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371-386, 2011.

[109] J. McDermott and C. Fox, "Using Abuse Case Models For Security Requirements Analysis," in *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC'99)*, pp. 55-64, 1999.

[110] G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80-83, 2004.

[111] G. McGraw, "Software Security: Building Security In," in *17th International Symposium on Software Reliability Engineering (ISSRE '06)*, pp. 6-6, 2006.

[112] G. McGraw and G. Morrisett, "Attacking Malicious Code: A Report to the Infosec Research Council," *IEEE Software*, vol. 17, no. 5, pp. 33-41, 2000.

[113] N. R. Mead and T. Stehney, *Security Quality Requirements Engineering*

*(SQUARE) Methodology*, vol. 30: ACM, 2005.

[114] Mehari, "Risk Analysis and Treatment Guide," CLUSIF, 2010.

[115] J. D. Meier, A. Mackman, S. Vasireddy, M. Dunner, R. Escamila and A. Murukan, *Improving Web Application Security: Threats and Countermeasures*, Microsoft, Available: http://www.cgisecurity.com/lib/Threats_Countermeasures.pdf, 2003.

[116] P. Mell, K. Scarfone and S. Romanosky, "CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0," in *FIRST-Forum of Incident Response and Security Teams*, pp. 1-23, 2007.

[117] D. Mellado, E. Fernández-Medina and M. Piattini, "Applying a Security Requirements Engineering Process," in *Computer Security–ESORICS 2006*, ed: Springer, pp. 192-206, 2006.

[118] S. J. Mellor, *MDA Distilled: Principles of Model-Driven Architecture*: Addison-Wesley Professional, 2004.

[119] S. J. Mellor, A. N. Clark and T. Futagami, "Model-Driven Development - Guest Editor's Introduction," *IEEE Software*, vol. 20, no. 5, pp. 14-18, 2003.

[120] T. O. Meservy and K. D. Fenstermacher, "Transforming Software Development: An MDA Road Map," *Computer*, vol. 38, no. 9, pp. 52-58, 2005.

[121] Microsoft, "The Security Risk Management Guide," Microsoft, 2006.

[122] C. Miller, "Security Considerations in Managing COTS Software," Available: https://buildsecurityin.us-cert.gov/articles/best-practices/legacy-systems/security-considerations-in-managing-cots-software, 2006.

[123] B. S. Mitchell and S. Mancoridis, "Comparing the Decompositions Produced by Software Clustering Algorithms Using Similarity Measurements," in *Proceedings of IEEE International Conference on Software Maintenance*, Florence, pp. 744-753, 2001.

[124] N. Moebius, K. Stenzel, H. Grandy and W. Reif, "SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications," in *International Conference on Availability, Reliability and Security (ARES '09)*, Fukuoka, pp.

841-846, 2009.

[125] A. Mourad, A. Soeanu, M.-A. Laverdière and M. Debbabi, "New Aspect-Oriented Constructs for Security Hardening Concerns," *Computers & Security,* vol. 28, no. 6, pp. 341-358, 2009.

[126] H. Mouratidis and P. Giorgini, "Secure Tropos: A Security-Oriented Extension of the Tropos Methodology," *International Journal of Software Engineering and Knowledge Engineering,* vol. 17, no. 02, pp. 285-309, 2007.

[127] H. Mouratidis, P. Giorgini and G. Manson, "Modelling Secure Multiagent Systems," in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Australia, pp. 859-866, 2003.

[128] H. Mouratidis, P. Giorgini, M. Schumacher and M. Manson, "Security Patterns for Agent Systems," in *Proceedings of the Eight European Conference on Pattern Languages of Programs (EuroPLoP)*, Irsee, Germany, 2003.

[129] S. Myagmar, A. J. Lee and W. Yurcik, "Threat Modeling as A Basis for Security Requirements," in *Symposium on Requirements Engineering for Information Security (SREIS)*, Paris, France, 2005.

[130] A. Nhlabatsi, B. Nuseibeh and Y. Yu, "Security Requirements Engineering for Evolving Software Systems: A Survey," *International Journal of Secure Software Engineering,* vol. 1, no. 1, pp. 54-73, 2010.

[131] NIST, Recommendation for Key Management-Part 1: General, National Institute of Standards and Technology, 2002.

[132] Nstalker, Available: http://www.nstalker.com/products/editions/free/, May 2012.

[133] OMG, "XML Metadata Interchange (XMI), v2.0 Specification," 2003.

[134] OMG, "UML 2.0 Superstructure Specification," 2004.

[135] OMG, "Meta Object Facility (MOF) Specification v1.4," 2002.

[136] OMG, "MOF 2.0 Query/ Views/ Transformations RFP, ad/2002-04-10," 2002.

[137] OMG, "MDA Guide Version 1.0.1," 2003.

[138] OWASP, "Comprehensive, Lightweight Application Security Process," Available: https://www.owasp.org/index.php/Category:OWASP_CLASP_Project, 2006.

[139] OWASP, "Threat Risk Modelling," Available: https://www.owasp.org/index.php/Threat_Risk_Modelling, Mar. 2010.

[140] Protégé, Available: http://protege.stanford.edu/, Nov. 2013.

[141] J. Pu, "Software Evolution through UML-Models Extraction," Ph.D thesis, De Montfort University, 2008.

[142] V. Raskin, C. F. Hempelmann, K. E. Triezenberg and S. Nirenburg, "Ontology in Information Security: A Useful Theoretical Foundation and Methodological Tool," in *Proceedings of the 2001 Workshop on New Security Paradigms*, Cloudcroft, New Mexico, pp. 53-59, 2001.

[143] J. Ren and R. Taylor, "A Secure Software Architecture Description Language," in *Workshop on Software Security Assurance Tools, Techniques, and Metrics*, USA, pp. 82-89, 2005.

[144] J. Reznik, T. Ritter, R. Schreiner and U. Lang, "Model Driven Development of Security Aspects," *Electronic Notes in Theoretical Computer Science*, vol. 163, no. 2, pp. 65-79, 2007.

[145] S. Romanosky, "Security Design Patterns," Available: http://www.romanosky.net/papers/securityDesignPatterns.html, 2002.

[146] M. Romero, B. D., H. M. Haddad and A. Molero, J. E., "A Methodological Tool for Asset Identification in Web Applications: Security Risk Assessment," in *Fourth International Conference on Software Engineering Advances (ICSEA '09)*, Porto, pp. 413-418, 2009.

[147] K. Saleh and M. Habil, "The Security Requirements Behavior Model for Trustworthy Software," in *2008 International MCETECH Conference on e-Technologies*, Montreal, Canada, pp. 235-238, 2008.

[148] P. Samuel and R. Mall, "Slicing-based Test Case Generation from UML Activity Diagrams," *ACM SIGSOFT Software Engineering Notes*, vol. 34, no. 6,

pp. 1-14, 2009.

[149] SANS, "A Security Checklist for Web Application Design," SANS Institute InfoSec Reading Room, Available: http://www.sans.org/reading-room/whitepapers/securecode/security-checklist-web-application-design-1389, 2004.

[150] M. Schumacher, *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications* vol. 2754: Springer-Verlag Berlin Heidelberg, 2003.

[151] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann and P. Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*: John Wiley & Sons, 2006.

[152] R. Sethi, "The Importance of Application Classification in Secure Application Development," Available: http://www.webappsec.org/projects/articles/041607.shtml, 2007.

[153] M. E. Shin and H. Gomaa, "Software Requirements and Architecture Modeling for Evolving Non-Secure Applications into Secure Applications," *Science of Computer Programming,* vol. 66, no. 1, pp. 60-70, 2007.

[154] G. Sindre and A. L. Opdahl, "Eliciting Security Requirements with Misuse Cases," *Requirements Engineering,* vol. 10, no. 1, pp. 34-44, 2005.

[155] R. Singh and V. Arora, "Literature Analysis on Model based Slicing," *International Journal of Computer Applications,* vol. 70, no. 16, pp. 45-51, 2013.

[156] E. Soler, J. Trujillo, C. Blanco and E. Fernández-Medina, "Designing Secure Data Warehouses by Using MDA and QVT," *Journal of Universal Computer Science,* vol. 15, no. 8, pp. 1607-1641, 2009.

[157] Sparx, *UML 2 Use Case Diagram,* Sparx systems, Available: http://www.sparxsystems.com/resources/uml2_tutorial/uml2_usecasediagram.html, Jul. 2013.

[158] C. Steel, R. Nagappan and R. Lai, *Core Security Patterns: Best Practices and*

*Strategies for J2EE, Web Services, and Identity Management*: Prentice-Hall, 2005.

[159] G. Stoneburner, A. Goguen and A. Feringa, "Risk Management Guide for Information Technology Systems," *NIST Special Publication*, vol. 800, no. 30, pp. 800-30, 2002.

[160] F. Swiderski and W. Snyder, *Threat Modeling*: Microsoft Press, 2009.

[161] F. Tip, "A Survey of Program Slicing Techniques," *Journal of Programming Languages*, vol. 3, no. 3, pp. 121-189, 1995.

[162] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The Knowledge Engineering Review*, vol. 11, no. 02, pp. 93-136, 1996.

[163] A. Van Lamsweerde, "From System Goals to Software Architecture," in *Formal Methods for Software Architectures*, ed: Springer, pp. 25-43, 2003.

[164] A. Van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proceedings of the 26th International Conference on Software Engineering*, pp. 148-157, 2004.

[165] A. Van Lamsweerde, R. Darimont and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 11, pp. 908-926, 1998.

[166] S. Van Langenhove, "Internal Broadcasting to Slice UML State Charts: as Rich as Needed," in *Proceedings of Abstracts of the FNRS Contact Day: The Theory and Practice of Software Verification*, 2005.

[167] M. VanHilst, E. B. Fernandez and F. Braz, "A Multi-Dimensional Classification for Users of Security Patterns," *Journal of Research & Practice in Information Technology*, vol. 41, no. 2, pp. 87-98, 2009.

[168] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*: Addison-Wesley Professional, 2001.

[169] VP, *Visual Paradigm* Available: http://www.visual-paradigm.com/, Nov. 2013.

[170] G. H. Walton, T. A. Longstaff and R. C. Linger, "Computational Evaluation of

Software Security Attributes," in *42nd Hawaii International Conference on System Sciences (HICSS'09)*, Big Island, pp. 1-10, 2009.

[171] J. Wang, W. Dong and Z. Qi, "Slicing Hierarchical Automata for Model Checking UML Statecharts," in *Proceedings of Fourth International Conference of Formal Engineering Methods*, Shanghai, China, pp. 435-446, 2002.

[172] M. S. Ware, J. B. Bowles and C. M. Eastman, "Using the Common Criteria to Elicit Security Requirements with Use Cases," in *Proceedings of the IEEE SoutheastCon*, pp. 273-278, 2005.

[173] I. Warren and J. Ransom, "Renaissance: A Method to Support Software System Evolution," in *Proceedings. 26th Annual International Computer Software and Applications Conference(COMPSAC'02)*, Secaucus, USA, pp. 415-420, 2002.

[174] WebStore, Available: http://sourceforge.net/projects/webstore-app/, Nov. 2013.

[175] M. Weiser, "Program Slicing," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 352-357, 1984.

[176] M. Weiser, "Program Slicing," in *Proceedings of the 5th International Conference on Software Engineering*, pp. 439-449, 1981.

[177] M. Weiss, "Patterns for Web Applications," in *Proceedings of the 10th conference on pattern languages of programming (PLoP '03)*, USA, 2003.

[178] M. Weiss, "Modeling security patterns using NFR analysis," *Integrating Security and Software Engineering: Advances and Future Visions*, pp. 127-141, 2007.

[179] I. S. Welch and R. J. Stroud, "Re-engineering Security as a Crosscutting Concern," *The Computer Journal*, vol. 46, no. 5, pp. 578-589, 2003.

[180] Wikipedia, *Visual Paradigm for UML*, Available: http://en.wikipedia.org/wiki/Visual_Paradigm_for_UML, 2010.

[181] F. Wu and T. Yi, "Dependence Analysis for UML Class Diagrams," *Journal of Electronics (China)*, vol. 21, no. 3, pp. 249-254, 2004.

[182] D. Xu, V. Goel and K. Nygard, "An Aspect-Oriented Approach to Security Requirements Analysis," in *30th Annual International Computer Software and*

*Applications Conference (COMPSAC'06)*, Chicago, USA, pp. 79-82, 2006.

[183] H. Yang and M. Ward, *Successful Evolution of Software Systems*: Artech House, 2003.

[184] J. Yoder and J. Barcalow, "Architectural Patterns for Enabling Application Security," in *Proceedings of the 4th Conference on Pattern Languages of Programming (PLoP '97)*, USA, 1998.

[185] N. Yoshioka, H. Washizaki and K. Maruyama, "A Survey on Security Patterns," *Progress in Informatics,* vol. 5, no. 5, pp. 35-47, 2008.

[186] K. Yskout, R. Scandariato, B. De Win and W. Joosen, "Transforming Security Requirements into Architecture," in *Third International Conference on Availability, Reliability and Security(ARES 08)*, Barcelona, pp. 1421-1428, 2008.

[187] J. Zhao, "Applying Slicing Technique to Software Architectures," in *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'98)*, Monterey, pp. 87-98, 1998.

[188] J. Zhao, "Using dependence analysis to support software architecture understanding," *New Technologies on Computer Software,* pp. 135-142, 2001.

[189] Z. J. Zhu and M. Zulkernine, "A Model-based Aspect-Oriented Framework for Building Intrusion-Aware Software Systems," *Information and Software Technology,* vol. 51, no. 5, pp. 865-875, 2009.

# Appendix A: Security Patterns Repository Organised by the Proposed Classification

| No. | **P1** |
|---|---|
| Name | **Audit Interceptor [158]** |
| Context | Application, Design |
| Problem | Deviations must be identified from the audit reports and corrective actions have to be taken so that the deviations do not recur, either through code fixes or policy changes. How can you make an auditing framework to easily support additions or changes to the auditing events? |
| Classification Key | Core Security, Repudiation |
| Solution | Intercept business tier requests and responses. Create audit events based on the information in the request response pair using declarative mechanisms defined externally to the application. The declarative approach is crucial to maintainability of the application. This makes it easy to keep up with the changed corporate policies. |
| **Related Patterns** | |
| No. | **P2** |
| Name | **Authenticator [151]** |
| Context | Application, Design |
| Problem | A malicious attacker might try to impersonate a legitimate user to have access to the user's resources. This could be even more serious if the impersonated user has a high level of privilege. |
| Classification Key | Perimeter Security, Spoofing, Authentication |
| Solution | Create a single point of access to receive the interactions of a subject and apply a protocol to verify the identity of the subject. Create a proof of identity if subject is successfully authenticated. |
| **Related Patterns** | Single Access Point, Policy Enforcement Point |
| No. | **P3** |
| Name | **Authentication Enforcer [158]** |
| Context | Application, Design |

| | |
|---|---|
| Problem | A malicious attacker might try to impersonate a legitimate user to have access to the user's resources. This could be even more serious if the impersonated user has a high level of privilege. How to prevent agents who are not allowed from entering the system? |
| Classification Key | Perimeter Security, Spoofing, Authentication, Web and J2EE |
| Solution | Create a single point of access to receive the interactions of a subject and apply a protocol to verify the identity of the subject. Create a proof of identity if subject is successfully authenticated. |
| **Related Patterns** | Single Access Point, Policy Enforcement Point |
| No. | **P4** |
| Name | **Authorisation [151]** |
| Context | Application, Architecture |
| Problem | Authentication is the process to verify the digital identity of the sender. Authorisation is the process that performs access control by deciding whether a program or a person has the privilege to access some data, functionality or service. Controlling access to the system resources and especially data is a key requirement for an application security. |
| Classification Key | Perimeter Security, Information Disclosure, Access Control, |
| Solution | Indicate, in a suitable representation, who is authorised to access what and in what way. Specify policies to define all the needed access to resources. |
| **Related Patterns** | Role Based Access Control, Reference Monitor |
| No. | **P5** |
| Name | **Authorisation Enforcer [158]** |
| Context | Application, Architecture |
| Problem | Authentication is the process to verify the digital identity of the sender. Authorisation is the process that performs access control by deciding whether a program or a person has the privilege to access some data, functionality or service. Controlling access to the system resources and especially data is a key requirement for an application security. How do we specify who is authorised to access specific resources in a system? |
| Classification Key | Perimeter Security, Information Disclosure, Access Control, Web and J2EE |
| Solution | Indicate, in a suitable representation, who is authorised to access what and in what way. Specify policies to define all the needed access to resources. |
| **Related Patterns** | Role Based Access Control, Reference Monitor |
| No. | **P6** |

## Appendix A: Security Patterns Repository Organised by the Proposed Classification

| | |
|---|---|
| Name | **Checkpointed System [16]** |
| Context | Application, Architecture |
| Problem | A component failure can result in loss or corruption of state information maintained by the failed component. Systems which rely on retained state for correct operation must be able to recover from loss or corruption of state information |
| Classification Key | Core Security, Tampering, Availability, |
| Solution | Create a set of states and make the system follow the state sequences in its life cycle. Store persistent state information all the time. Use a wide variety of configurations that provide the ability to restart the system from a known valid state (i.e. the checkpoint), either on the same platform or on different platforms. |
| No. | **P7** |
| Name | **Comparator Checked Fault Tolerant System [16]** |
| Context | Application, Architecture |
| Problem | Failure in one component often starts the domino-effect and the failure is propagated to bring about the entire system failure. |
| Classification Key | Exterior Security, Tampering, Availability |
| Solution | Structure a system so that an independent failure of one component will be detected quickly and so that an independent single-component failure will not cause system failure. Use multiple components to perform the tasks and report the result only if the result is similar for both the components. |
| **Related Patterns** | Encrypted Storage |
| **No.** | **P8** |
| **Name** | **Container Managed Security [158]** |
| Context | Application, Design |
| Problem | Adding programmatic security solutions to an application involves extra work on development of security libraries and verification of the implementation. For many applications, the choice would be to use declarative security. How can security be added declaratively to an application? |
| Classification Key | Core Security, Information Disclosure, Authentication, Access Control |
| Solution | Use standard security features provided by application container. Define application level roles at development time. Perform a mapping of these application level logical roles to users in the deployment environment at deployment time or thereafter. |
| **Related Patterns** | Intercepting Web Agent |

| No. | **P9** |
|---|---|
| Name | **DoS Safety [67]** |
| Context | Application, Design |
| Problem | Denial of Service is tackled by adopting several network-based strategies. However, system architecture should be resilient to such attacks as well. |
| Classification Key | Core Security, DoS, Availability, |
| Solution | Protect against Denial of Service attacks by setting resource limit. This can be done by per process resource management or by adopting operating system's resource management features. |
| **Related Patterns** | Small Processes |
| No. | **P10** |
| Name | **Encrypted Storage [91]** |
| Context | Application, Architecture |
| Problem | Firewalls provide protection of data stored in a server by limiting access to it. However, data can still be accessed by hackers. The extreme option to protect data in server is not to store any sensitive information at all in server. This is infeasible, because the server needs to keep state. Since firewalls do not provide security at data level, there is need of some additional security in this level. |
| Classification Key | Core Security, Information Disclosure, Confidentiality |
| Solution | Encrypt critical data before storing them in the server. Decrypt data in memory before they are used by the server. Use a single key for encrypting all the data and periodically alter it if possible. This involves decrypting all data stored with previous key and re-encrypting it with the new key. If this leads to an availability problem, use a large encryption key that is difficult to compromise. |
| **Related Patterns** | Encrypted Storage |
| No. | **P11** |
| Name | **Error Detection and Correction [16]** |
| Context | Application, Architecture |
| Problem | Data residing on storage media or in transit across communication links is often susceptible to errors. This can occur because of bugs in the local system, or because of active attempts by adversaries to corrupt the data. |
| Classification Key | Core Security, Tampering, Integrity |
| Solution | Add redundancy to data to facilitate detection of and recovery from errors. This can be done by simple |

| | |
|---|---|
| | parity checking or integrity checking based on some hash values |
| **Related Patterns** | Comparator Checked Fault Tolerant System. |
| No. | **P12** |
| Name | **Full Access with Errors [151]** |
| Context | Application, Architecture |
| Problem | Some user interfaces offer different options based on the privilege level of the user. The designer of such a user interface faces the challenge that the revelation of the complete interface can cause a problem because the user may not have rights to invoke all functionality. Even the access rights might not be known in advance. This problem generalises to any interface you design whenever there are multiple modes of usage, such as different access rights. |
| Classification Key | Core Security, Information Disclosure, Access Control, Authentication |
| Solution | Design the application so users see everything that is available to them. When a user performs an operation, check if it is allowed. Generate error notifications if they try to access unauthorised operations. |
| **Related Patterns** | Limited Access, Policy Enforcement Point, Security Session |
| No. | **P13** |
| Name | **Intercepting Validator [158]** |
| Context | Data, Design |
| Problem | Several well-known attack strategies involve compromising a system by sending requests with invalid data or malicious code. This entails injection of malicious scripts, SQL statements, XML content and invalid data. These attacks can be avoided by validating data before use. Because of the constantly changing attack patterns, the data validation mechanism has to continuously change to prevent against new attacks. Another concern is the freshness of data. An application cannot blindly trust the freshness of data |
| Classification Key | Core Security, Spoofing, Filtering, Integrity, Web and J2EE |
| Solution | Verify the user input before they are used. Use a pluggable filters approach and apply the filters declaratively based on URL, allowing different requests to be mapped to different filter chains. Restrict filter tasks to pre-processing of requests and providing validation, i.e. a yes or no decision. Apply validation in the server side, because client side validation is insecure and open to spoofing. Renegotiate trust between users from time to time. Keep a record of the volatility of the data. |
| **Related Patterns** | Session |
| No. | **P14** |
| Name | **Intercepting Web Agent [158]** |

| | |
|---|---|
| Context | Application, Design |
| Problem | Security is often postponed until after the functional pieces of the application have been designed. After an application is deployed, it is very difficult to implement the authentication, authorisation and auditing mechanism. |
| Classification Key | Perimeter Security, Information Disclosure, Access Control, Authentication |
| Solution | Provide authentication and authorisation outside the application. Use an intercepting agent installed on web server and provide authentication and authorisation of incoming requests by intercepting them and enforcing access control policy at the web server. Isolate application logic from security logic. |
| **Related Patterns** | Secure Service Proxy |
| No. | **P15** |
| Name | **Limited Access [151]** |
| Context | Application, Architecture |
| Problem | Presenting the entire user interface has an important security problem associated with it. Some options may be private for some privileged user group and other users should not even see those options. Seeing the entire user interface is annoying for a user who has access to only a few operations when he finds by clicking options that he is not entitled to perform those operations. |
| Classification Key | Core Security, Information Disclosure, Access Control, Authentication |
| Solution | Only let the users see what they have access to. Only give them selections and menus to options that their current access-privileges permit. Dynamically adjust the view when the permissions of the user change |
| **Related Patterns** | Full Access with Errors, Policy Enforcement Point, Security Session, chroot Jail |
| No. | **P16** |
| Name | **Multilevel Security[151]** |
| Context | Application, Architecture |
| Problem | How do you control access in an environment with sensitive data so as to prevent leakage of information |
| Classification Key | Core Security, Access Control |
| Solution | Assign classification to users and data. Separate different institutional units into categories. Enforce confidentiality and integrity by adopting security models. For example, confidentiality can be enforced by the Bell La-Padula model and integrity can be enforced by Biba's model. |
| **Related Patterns** | Role Based Access Control. |
| No. | **P17** |

| | |
|---|---|
| Name | **Policy[16]** |
| Context | Application, Architecture |
| Problem | Systems often express their requirements as policies. However, the policy has to be enforced to check conformance. The policy enforcement functions have to be invoked in a correct sequence. |
| Classification Key | Perimeter Security, Access Control |
| Solution | Isolate the part that makes policy enforcement decision in a discrete component of the system. Ensure that policy enforcement activities are performed in proper sequence |
| **Related Patterns** | Role Based Access Control, Policy Enforcement Point |
| No. | **P18** |
| Name | **Protected System [16]** |
| Context | ALL, Architecture |
| Problem | Protecting systems against unauthorised access is the first line of defence. At the entry point, the requests have to evaluate and access permission is granted only if they conform to some policy. |
| Classification Key | Security Pattern Space, Access Control |
| Solution | Structure a system so that all access by clients to resources is mediated by a guard which enforces a security policy. The guard can be any type of firewall that acts as the policy enforcement point. Each resource or the overall system is protected by a guard that evaluates the incoming requests |
| **Related Patterns** | Single Access Point, Policy Enforcement Point, Policy. |
| No. | **P19** |
| Name | **Reference Monitor [151]** |
| Context | Application, Architecture |
| Problem | Authorisation policies have to be enforced. How can the authorisation policies be enforced to prevent the users and processes from performing illegal actions ? |
| Classification Key | Core Security, Information Disclosure, Access Control |
| Solution | Define a process that intercepts all requests for resources and validates access on them. |
| **Related Patterns** | Policy Enforcement Point |
| No. | **P20** |
| Name | **Replicated System [16]** |

| | |
|---|---|
| Context | Application, Architecture |
| Problem | Transactional systems are often susceptible to outages of communication links, communication elements or other system elements. It is important to assure availability of transactional services in the midst of such failures. |
| Classification Key | Exterior Security, Tampering, Availability |
| Solution | Replicate services at multiple points in the network and during failure replace the failed service with an available service. Perform load-balancing based on some workload scheduling mechanism by a workload management proxy |
| **Related Patterns** | Standby |
| No. | **P21** |
| Name | **Role Based Access Control [151]** |
| Context | Application, Architecture |
| Problem | Specifying Authorisation policies becomes difficult if a system has many users and resources. How do we reduce the number of individual rights when there are many subjects and objects involved |
| Classification Key | Perimeter Security, Information Disclosure, Access Control |
| Solution | Group subjects into roles based on similarities in duties performed. Assign rights of accessing objects to roles. |
| **Related Patterns** | Authorisation |
| No. | **P22** |
| Name | **Secure Base Action [158]** |
| Context | All, Design |
| Problem | Security-related data and methods are used across many or most of the Web tier components. Operations such as verifying authentication, checking authorisation, and storing and retrieving session information are prevalent throughout the servlets and JSPs. Due to the nature of security, these operations are often tied together through their implementation. When many normal application components are exposed to many of these security-related components, flexibility and reuse are reduced because of the inherent underlying coupling of security components. |
| Classification Key | Perimeter Security, pattern space, Authentication, Access Control, Web and J2EE |
| Solution | A Secure Base Action pattern can be used as a single point for security-related functionality within the Web tier. By having Web components such as Front Controllers and Application Controllers inherit from it, they gain access to all of the security operations that are necessary throughout the front end. Authentication, authorisation, validation, logging, and session management are areas that the Secure Base Action |

| | |
|---|---|
| | encapsulates and provides centralised access to. |
| **Related Patterns** | Intercepting Validator, Secure Logger, Authorisation Enforcer, Authentication Enforcer |
| No. | **P23** |
| Name | **Secure Communication [16]** |
| Context | Network, Architecture |
| Problem | The communication channel is susceptible to various attacks and it is inherently unreliable. The communication between two protected systems or a subject and a protected system is jeopardised by the unreliability of the communication channel |
| Classification Key | Exterior Security, Information Disclosure, Confidentiality |
| Solution | Ensure that mutual security policy objectives are met when there is need for two parties to communicate in the presence of threats. Create secure channel for sensitive data that obscure the data in transit. Reduce the associated overhead on the system by using ordinary communication channels for non-sensitive data. |
| **Related Patterns** | Protected System, Security Association, Information Obscurity |
| No. | **P24** |
| Name | **Secure Logger [158]** |
| Context | Data, Design |
| Problem | Application Logs have to be created appropriately at multiple points during an application's operational life cycle. Event logs and related data must be secured against alteration by an attacker. Log data should not be accessible to unauthorised personnel. |
| Classification Key | Exterior Security, Tampering, Accountability, Non-repudiation, Web and J2EE |
| Solution | Use a centrally controlled logging functionality that can be used in various places throughout the application request and response. Decouple the logging functionality and provide it as a component or service to be used throughout the application. Cryptographically secure the logged data and keep additional information to verify the integrity of logged data. Control access to the log so that unauthorised users cannot view content. |
| **Related Patterns** | Standby |
| No. | **P25** |
| Name | **Secure Pipe [158]** |
| Context | Network, Design |
| Problem | Web-based transactions are often exposed to eavesdropping, replay, and spoofing attacks. Anytime a request goes over an insecure network, the data can be intercepted or exposed by unauthorised users. Even within |

| | |
|---|---|
| | the confines of a VPN, data is exposed at the endpoint, such as inside of an intranet. When exposed, it is subject to disclosure, modification, or duplication |
| Classification Key | Network layer, Exterior Security, Web and J2EE, Information Disclosure |
| Solution | A Secure Pipe provides a simple and standardised way to protect data sent across a network. It does not require application-layer logic and therefore reduces the complexity of implementation. In some instances, the task of securing the pipe can actually be moved out of the application and even off of the hardware platform altogether. |
| **Related Patterns** | Point-to-Point Channel |
| No. | **P26** |
| Name | **Secure Service Facade [158]** |
| Context | Application, Design |
| Problem | Many access points in the business tier mean that many points of failure that have to be secured. Every access point has to have authentication and authorisation and data validation and auditing mechanism. This becomes an even more difficult problem if security has to be retrofitted. How can we provide a secure interface for a fine-grained and loosely coupled security service? |
| Classification Key | Core Security, Web and J2EE |
| Solution | Integrate fine-grained, security unaware service implementation into a unified, security-enabled interface to clients. Use it as a gateway where client requests are securely validated and routed to the appropriate fine-grained service implementation. Maintain and mediate the security and workflow context between interactive client requests and fine-grained services that fulfil portions of client requests. |
| **Related Patterns** | Intercepting Web Agent, Secure Message Router |
| No. | **P27** |
| Name | **Secure Service Proxy [158]** |
| Context | All , Design |
| Problem | Adaptation of existing systems to newer security protocols is a standard practice in software maintenance. In case of SOA, you want to expose your existing system as services that interact with other services, but their security protocols do not match |
| Classification Key | Core Security, Authentication, Access Control, Web and J2EE |
| Solution | Provide security service as a wrapper. Intercept all the requests from clients, identify the requested service, enforce the security policy as required by the service, optionally transform the request for the inbound protocol to that expected by the service, and finally forward the request to the appropriate destination service. On the return path, transform the results according to outbound requirements. Externalise the addition of security logic to existing applications. |

| | |
|---|---|
| **Related Patterns** | Intercepting Web Agent, Secure Message Router |
| No. | **P28** |
| Name | **Security Association [16]** |
| Context | Application, Architecture |
| Problem | Secure Communication pattern adds overhead because it adds expensive security mechanisms. It is better if the security associated information is not added to the data content every time two parties communicate but only used when the connection is established. This requires storing security related information at each end of communications channel. |
| Classification Key | Perimeter Security, Information Disclosure, Authentication |
| Solution | Define a structure that provides each participant in a secure communication with the information it will use to protect messages to be transmitted to the other party, and with the information which it will use to understand and verify the protection applied to messages received from the other party. |
| **Related Patterns** | Secure Communication |
| No. | **P29** |
| Name | **Security Context [16]** |
| Context | Application, Architecture |
| Problem | Within a single execution context, program or processes need to act on behalf of multiple subjects. When an execution context, program or process needs to act on behalf of a single subject on multiple occasions over a period of time, it needs to be able to have access to information about the subject whenever it needs to take an action. |
| Classification Key | Core Security, Elevation of Privilege, Confidentiality |
| Solution | Provide a container for security attributes and data relating to a particular execution context, process, operation or action. |
| **Related Patterns** | Security Association, Subject Descriptor |
| No. | **P30** |
| Name | **Security Session [151]** |
| Context | Application, Architecture |
| Problem | Many web based transactions require the user to browse through multiple web pages. Normally the user logs in at the start of transaction and then follows multiple web pages. Different components acting on behalf of a user might need to know, which user is activating them and what are the user's permissions. Having every individual component or program within the system identifying, authenticating and authorising users is |

| | |
|---|---|
| | annoying to both users and developers. In addition, system components might call each other or work together and thus need a way to share information regarding the user without compromising this 'global' data to other users |
| Classification Key | Core Security, Information Disclosure, Integrity |
| Solution | Create a session object, that holds all of the variables that need to be shared by many objects. Associate every action of the user with the session. |
| **Related Patterns** | Single Sign On, Policy Enforcement Point, Integration Reverse Proxy, Front Door. |
| No. | **P31** |
| Name | **Single Access Point [151]** |
| Context | Application, Architecture |
| Problem | A security model is difficult to validate when there are multiple ways for entering the application. How can we secure a system from outside intrusion? |
| Classification Key | Perimeter Security, Information Disclosure, Authentication |
| Solution | Set up only one way to get into the system and if necessary, create a mechanism to decide which sub-application to launch. Typically most applications use a log in screen to accomplish the single access point |
| **Related Patterns** | Policy Enforcement Point, Security Session |
| No. | **P32** |
| Name | **Standby [16]** |
| Context | Application, Architecture |
| Problem | A system component is exposed to failure. Failure of a single component might cause a system outage. How can an available system be designed where a system is tolerant of component failure? |
| Classification Key | Exterior Security, DoS, Availability |
| Solution | Structure a system with backup components so that the service provided by one component can be resumed by the backup component in case of system failure. In many systems, it is cost-effective to have a backup recovery mechanism |
| **Related Patterns** | Replicated System |

# Appendix B: OWL Representation of the Proposed Security Ontology

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY Security "http://www.semanticweb.org/guah1/ontologies/Security.owl#" >
    <!ENTITY untitled-ontology-12 "http://www.semanticweb.org/guah1/ontologies/2014/0/untitled-ontology-12#" >
]>

<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.w3.org/2002/07/owl"
     xmlns:Security="http://www.semanticweb.org/guah1/ontologies/Security.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:untitled-ontology-12="http://www.semanticweb.org/guah1/ontologies/2014/0/untitled-ontology-12#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <Ontology rdf:about="http://www.semanticweb.org/guah1/ontologies/Security.owl">
        <rdfs:comment rdf:datatype="&xsd;string">An ontology that describes security requirements,security patterns and their
relationships. Typical starting points for browsing should be the classes Asset, Threat, SecurityPattern.</rdfs:comment>
        <Security:Creator rdf:datatype="&xsd;string">Hui Guan, guanh1999@126.com</Security:Creator>
    </Ontology>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Annotation properties
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Creator -->

    <AnnotationProperty rdf:about="&Security;Creator"/>

   <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#containsThreat -->

    <ObjectProperty rdf:about="&Security;containsThreat">
        <rdfs:domain rdf:resource="&Security;Layer"/>
        <rdfs:range rdf:resource="&Security;Threat"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasApplicationContext -->

    <ObjectProperty rdf:about="&Security;hasApplicationContext">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&Security;AppicationContext"/>
```

```
        <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasAsset -->

    <ObjectProperty rdf:about="&Security;hasAsset"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasDomain -->

    <ObjectProperty rdf:about="&Security;hasDomain">
        <rdfs:range rdf:resource="&Security;DomainSpecific"/>
        <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasLayer -->

    <ObjectProperty rdf:about="&Security;hasLayer"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasLifecycle -->

    <ObjectProperty rdf:about="&Security;hasLifecycle"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasProblem -->

    <ObjectProperty rdf:about="&Security;hasProblem"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSecurityAttribute -->

    <ObjectProperty rdf:about="&Security;hasSecurityAttribute">
        <rdfs:domain rdf:resource="&Security;Asset"/>
        <rdfs:range rdf:resource="&Security;SecurityAttribute"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSecurityConcerns -->

    <ObjectProperty rdf:about="&Security;hasSecurityConcerns"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSolutionType -->

    <ObjectProperty rdf:about="&Security;hasSolutionType">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="&Security;SecuritySolution"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasThreat -->

    <ObjectProperty rdf:about="&Security;hasThreat">
        <rdfs:subPropertyOf rdf:resource="&Security;hasProblem"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasThreatType -->

    <ObjectProperty rdf:about="&Security;hasThreatType">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
        <rdfs:range rdf:resource="&Security;ThreatType"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#isSolvedBy -->

    <ObjectProperty rdf:about="&Security;isSolvedBy">
```

```
        <rdfs:range rdf:resource="&Security;SecurityPattern"/>
        <rdfs:domain rdf:resource="&Security;Threat"/>
        <inverseOf rdf:resource="&Security;hasThreat"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#isSpecialisedBy -->

    <ObjectProperty rdf:about="&Security;isSpecialisedBy">
        <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
        <rdfs:range rdf:resource="&Security;SecurityPattern"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#isThreatenedBy -->

    <ObjectProperty rdf:about="&Security;isThreatenedBy">
        <rdfs:domain rdf:resource="&Security;Asset"/>
        <rdfs:range rdf:resource="&Security;SecurityProblem"/>
        <inverseOf rdf:resource="&Security;hasAsset"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#requires -->

    <ObjectProperty rdf:about="&Security;requires">
        <rdfs:domain rdf:resource="&Security;SecurityPattern"/>
        <rdfs:range rdf:resource="&Security;SecurityPattern"/>
    </ObjectProperty>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#residesOn -->

    <ObjectProperty rdf:about="&Security;residesOn">
        <rdfs:range rdf:resource="&Security;Layer"/>
        <rdfs:domain rdf:resource="&Security;Threat"/>
        <inverseOf rdf:resource="&Security;containsThreat"/>
    </ObjectProperty>


    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#hasSeverityScale -->

    <DatatypeProperty rdf:about="&Security;hasSeverityScale">
        <rdfs:domain rdf:resource="&Security;Threat"/>
        <rdfs:range rdf:resource="&owl;real"/>
    </DatatypeProperty>


    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AppicationContext -->

    <Class rdf:about="&Security;AppicationContext">
        <rdfs:subClassOf rdf:resource="&Security;ClassificationKey"/>
    </Class>
```

```
    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ApplicationLevel -->

    <Class rdf:about="&Security;ApplicationLevel">
        <rdfs:subClassOf rdf:resource="&Security;Threat"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ApplicationSoftware -->

    <Class rdf:about="&Security;ApplicationSoftware">
        <rdfs:subClassOf rdf:resource="&Security;Software"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ApplicationSpecific -->

    <Class rdf:about="&Security;ApplicationSpecific">
        <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Asset -->

    <Class rdf:about="&Security;Asset">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AuditingAndLogging -->

    <Class rdf:about="&Security;AuditingAndLogging">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>Who did what and when? Auditing and logging refer to how your
application records security-related events.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authentication -->

    <Class rdf:about="&Security;Authentication">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment xml:lang="en">Who are you? Authentication is the process that an entity uses to
identify another entity, typically through credentials such as a user name
and password.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authorisation -->

    <Class rdf:about="&Security;Authorisation">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>What can you do? Authorisation is the process that an application uses
to control access to resources and operations.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ClassificationKey -->

    <Class rdf:about="&Security;ClassificationKey">
        <rdfs:subClassOf rdf:resource="&Security;SecuritySolution"/>
        <rdfs:comment xml:lang="en">some key words to help the seucity pattern classification</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Communication -->

    <Class rdf:about="&Security;Communication">
        <rdfs:subClassOf rdf:resource="&Security;Physical"/>
    </Class>
```

```
    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ConfigurationManagement -->

    <Class rdf:about="&Security;ConfigurationManagement">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>Who does your application run as? Which databases does it connect to?
How is your application administered? How are these settings secured?
Configuration management refers to how your application handles these
operational issues.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Cryptography -->

    <Class rdf:about="&Security;Cryptography">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>How are you protecting secret information (confidentiality)? How are you
tamperproofing your data or libraries (integrity)? How are you providing
seeds for random values that must be cryptographically strong?
Cryptography refers to how your application enforces confidentiality and
integrity.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Data -->

    <Class rdf:about="&Security;Data">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DistribitedSystems -->

    <Class rdf:about="&Security;DistribitedSystems">
        <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Document -->

    <Class rdf:about="&Security;Document">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DomainSpecific -->

    <Class rdf:about="&Security;DomainSpecific">
        <rdfs:subClassOf rdf:resource="&Security;ClassificationKey"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#EmbeddedSystems -->

    <Class rdf:about="&Security;EmbeddedSystems">
        <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ExceptionManagement -->

    <Class rdf:about="&Security;ExceptionManagement">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>When a method call in your application fails, what does your application
do? How much does it reveal about the failure condition? Do you return
friendly error information to end users? Do you pass valuable exception
information back to the caller? Does your application fail gracefully?</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ExternalPersonnel -->

    <Class rdf:about="&Security;ExternalPersonnel">
```

```
        <rdfs:subClassOf rdf:resource="&Security;Personnel"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Hardware -->

    <Class rdf:about="&Security;Hardware">
        <rdfs:subClassOf rdf:resource="&Security;Physical"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#HostLevel -->

    <Class rdf:about="&Security;HostLevel">
        <rdfs:subClassOf rdf:resource="&Security;Threat"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#InputValidation -->

    <Class rdf:about="&Security;InputValidation">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>How do you know that the input your application receives is valid and
safe? Input validation refers to how your application filters, scrubs, or
rejects input before additional processing.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#InternalData -->

    <Class rdf:about="&Security;InternalData">
        <rdfs:subClassOf rdf:resource="&Security;Data"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#InternalPersonnel -->

    <Class rdf:about="&Security;InternalPersonnel">
        <rdfs:subClassOf rdf:resource="&Security;Personnel"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Layer -->

    <Class rdf:about="&Security;Layer">
        <rdfs:subClassOf rdf:resource="&Security;SecurityContext"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#LifeCycle -->

    <Class rdf:about="&Security;LifeCycle">
        <rdfs:subClassOf rdf:resource="&Security;SecurityContext"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Media -->

    <Class rdf:about="&Security;Media">
        <rdfs:subClassOf rdf:resource="&Security;Physical"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#NetworkLevel -->

    <Class rdf:about="&Security;NetworkLevel">
        <rdfs:subClassOf rdf:resource="&Security;Threat"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#OS -->
```

```
    <Class rdf:about="&Security;OS">
        <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ParameterManipulation -->

    <Class rdf:about="&Security;ParameterManipulation">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>Form fields, query string arguments, and cookie values are frequently
used as parameters for your application. Parameter manipulation refers
to both how your application safeguards tampering of these values and
how your application processes input parameters.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Personnel -->

    <Class rdf:about="&Security;Personnel">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Physical -->

    <Class rdf:about="&Security;Physical">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Priority -->

    <Class rdf:about="&Security;Priority">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#PublicData -->

    <Class rdf:about="&Security;PublicData">
        <rdfs:subClassOf rdf:resource="&Security;Data"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Reputation -->

    <Class rdf:about="&Security;Reputation">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SCADA -->

    <Class rdf:about="&Security;SCADA">
        <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityAttribute -->

    <Class rdf:about="&Security;SecurityAttribute">
        <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityConcern -->

    <Class rdf:about="&Security;SecurityConcern">
        <rdfs:subClassOf rdf:resource="&Security;ClassificationKey"/>
    </Class>
```

```
    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityContext -->

    <Class rdf:about="&Security;SecurityContext">
        <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityPattern -->

    <Class rdf:about="&Security;SecurityPattern">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:comment xml:lang="en">A Security Pattern is a description of one particular recurring security problem that arises
in specific contexts and presents a well-proven generic scheme for its solution.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityPatternElement -->

    <Class rdf:about="&Security;SecurityPatternElement"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityProblem -->

    <Class rdf:about="&Security;SecurityProblem">
        <equivalentClass rdf:resource="&Security;Threat"/>
        <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecurityRequirementElement -->

    <Class rdf:about="&Security;SecurityRequirementElement"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SecuritySolution -->

    <Class rdf:about="&Security;SecuritySolution">
        <rdfs:subClassOf rdf:resource="&Security;SecurityPatternElement"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SensitiveData -->

    <Class rdf:about="&Security;SensitiveData">
        <rdfs:subClassOf rdf:resource="&Security;Data"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Sensitive_Data -->

    <Class rdf:about="&Security;Sensitive_Data">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>Sensitive data is information that must be protected either in memory,
over the wire, or in persistent stores. Your application must have a
process for handling sensitive data.</rdfs:comment>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Service -->

    <Class rdf:about="&Security;Service">
        <rdfs:subClassOf rdf:resource="&Security;Asset"/>
    </Class>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SessionManagement -->

    <Class rdf:about="&Security;SessionManagement">
        <rdfs:subClassOf rdf:resource="&Security;ApplicationLevel"/>
        <rdfs:comment>A session refers to a series of related interactions between a user and
your Web application. Session management refers to how your
application handles and protects these interactions.</rdfs:comment>
```

```
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Software -->

<Class rdf:about="&Security;Software">
      <rdfs:subClassOf rdf:resource="&Security;Asset"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SolutionType -->

<Class rdf:about="&Security;SolutionType">
      <rdfs:subClassOf rdf:resource="&Security;SecuritySolution"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SystemSoftware -->

<Class rdf:about="&Security;SystemSoftware">
      <rdfs:subClassOf rdf:resource="&Security;Software"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Threat -->

<Class rdf:about="&Security;Threat">
      <rdfs:subClassOf rdf:resource="&Security;SecurityRequirementElement"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ThreatType -->

<Class rdf:about="&Security;ThreatType">
      <rdfs:subClassOf rdf:resource="&Security;ClassificationKey"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UbiquitousComputing -->

<Class rdf:about="&Security;UbiquitousComputing">
      <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
</Class>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#WebAndJ2EE -->

<Class rdf:about="&Security;WebAndJ2EE">
      <rdfs:subClassOf rdf:resource="&Security;DomainSpecific"/>
</Class>


<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Individuals
//
///////////////////////////////////////////////////////////////////////////////////
  -->


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AccessControl -->

<NamedIndividual rdf:about="&Security;AccessControl">
      <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AccessSensitiveDataInStorage -->

<NamedIndividual rdf:about="&Security;AccessSensitiveDataInStorage">
      <rdf:type rdf:resource="&Security;Sensitive_Data"/>
```

```
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Accountability -->

<NamedIndividual rdf:about="&Security;Accountability">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Analysis -->

<NamedIndividual rdf:about="&Security;Analysis">
    <rdf:type rdf:resource="&Security;LifeCycle"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Application -->

<NamedIndividual rdf:about="&Security;Application">
    <rdf:type rdf:resource="&Security;Layer"/>
    <Security:containsThreat rdf:resource="&Security;DataTampering"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ArbitaryCodeExecution -->

<NamedIndividual rdf:about="&Security;ArbitaryCodeExecution">
    <rdf:type rdf:resource="&Security;HostLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AttackerCovrsHisOrHerTracks -->

<NamedIndividual rdf:about="&Security;AttackerCovrsHisOrHerTracks">
    <rdf:type rdf:resource="&Security;AuditingAndLogging"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AttackerExploitsAnApplicationWithoutTrace -->

<NamedIndividual rdf:about="&Security;AttackerExploitsAnApplicationWithoutTrace">
    <rdf:type rdf:resource="&Security;AuditingAndLogging"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#AttackerRevealsImplementationDetails -->

<NamedIndividual rdf:about="&Security;AttackerRevealsImplementationDetails">
    <rdf:type rdf:resource="&Security;ExceptionManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authentication -->

<NamedIndividual rdf:about="&Security;Authentication">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
    <Security:hasLayer rdf:resource="&Security;Application"/>
    <Security:hasThreat rdf:resource="&Security;DataTampering"/>
    <Security:hasLifecycle rdf:resource="&Security;Design"/>
    <Security:hasSecurityAttribute rdf:resource="&Security;Integrity"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authenticator -->

<NamedIndividual rdf:about="&Security;Authenticator">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Authorisation -->
```

```
<NamedIndividual rdf:about="&Security;Authorisation">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Availability -->

<NamedIndividual rdf:about="&Security;Availability">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#BruteForceAttack -->

<NamedIndividual rdf:about="&Security;BruteForceAttack">
    <rdf:type rdf:resource="&Security;Authentication"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#BufferOverflow -->

<NamedIndividual rdf:about="&Security;BufferOverflow">
    <rdf:type rdf:resource="&Security;InputValidation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Canonicalisation -->

<NamedIndividual rdf:about="&Security;Canonicalisation">
    <rdf:type rdf:resource="&Security;InputValidation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CheckPointed_System -->

<NamedIndividual rdf:about="&Security;CheckPointed_System">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Check_Point -->

<NamedIndividual rdf:about="&Security;Check_Point"/>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ChecksumSpoofing -->

<NamedIndividual rdf:about="&Security;ChecksumSpoofing">
    <rdf:type rdf:resource="&Security;Cryptography"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Client_Input_Filter -->

<NamedIndividual rdf:about="&Security;Client_Input_Filter"/>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Comparator-checked_Fault_tolerant -->

<NamedIndividual rdf:about="&Security;Comparator-checked_Fault_tolerant">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Confidentiality -->

<NamedIndividual rdf:about="&Security;Confidentiality">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>
```

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CookieManipulation -->

<NamedIndividual rdf:about="&Security;CookieManipulation">
    <rdf:type rdf:resource="&Security;ParameterManipulation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CookieReplay -->

<NamedIndividual rdf:about="&Security;CookieReplay">
    <rdf:type rdf:resource="&Security;Authentication"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CoreSecurity -->

<NamedIndividual rdf:about="&Security;CoreSecurity">
    <rdf:type rdf:resource="&Security;AppicationContext"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CredentialTheft -->

<NamedIndividual rdf:about="&Security;CredentialTheft">
    <rdf:type rdf:resource="&Security;Authentication"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#CrossSiteScripting -->

<NamedIndividual rdf:about="&Security;CrossSiteScripting">
    <rdf:type rdf:resource="&Security;InputValidation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DOS_safety -->

<NamedIndividual rdf:about="&Security;DOS_safety">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DataTampering -->

<NamedIndividual rdf:about="&Security;DataTampering">
    <rdf:type rdf:resource="&Security;Authorisation"/>
    <rdf:type rdf:resource="&Security;Sensitive_Data"/>
    <Security:residesOn rdf:resource="&Security;Application"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DenialOfService -->

<NamedIndividual rdf:about="&Security;DenialOfService">
    <rdf:type rdf:resource="&Security;ExceptionManagement"/>
    <rdf:type rdf:resource="&Security;HostLevel"/>
    <rdf:type rdf:resource="&Security;NetworkLevel"/>
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Design -->

<NamedIndividual rdf:about="&Security;Design">
    <rdf:type rdf:resource="&Security;LifeCycle"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Detection -->

<NamedIndividual rdf:about="&Security;Detection">
    <rdf:type rdf:resource="&Security;SolutionType"/>
```

```
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Deterrence -->

<NamedIndividual rdf:about="&Security;Deterrence">
    <rdf:type rdf:resource="&Security;SolutionType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DictionaryAttack -->

<NamedIndividual rdf:about="&Security;DictionaryAttack">
    <rdf:type rdf:resource="&Security;Authentication"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DisclosureOfConfidentialData -->

<NamedIndividual rdf:about="&Security;DisclosureOfConfidentialData">
    <rdf:type rdf:resource="&Security;Authorisation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#DistributedSystems -->

<NamedIndividual rdf:about="&Security;DistributedSystems">
    <rdf:type rdf:resource="&Security;DomainSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ElevationOfPrivilege -->

<NamedIndividual rdf:about="&Security;ElevationOfPrivilege">
    <rdf:type rdf:resource="&Security;Authorisation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Elevationofprivilege -->

<NamedIndividual rdf:about="&Security;Elevationofprivilege">
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#EmbeddedSystems -->

<NamedIndividual rdf:about="&Security;EmbeddedSystems">
    <rdf:type rdf:resource="&Security;DomainSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Encrypted_storage -->

<NamedIndividual rdf:about="&Security;Encrypted_storage">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Error_detection/_correction -->

<NamedIndividual rdf:about="&Security;Error_detection/_correction">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ExteriorSecurity -->

<NamedIndividual rdf:about="&Security;ExteriorSecurity">
    <rdf:type rdf:resource="&Security;AppicationContext"/>
</NamedIndividual>
```

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Filtering -->

<NamedIndividual rdf:about="&Security;Filtering">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Firewall -->

<NamedIndividual rdf:about="&Security;Firewall">
    <rdf:type rdf:resource="&Security;ApplicationSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#FootPrinting -->

<NamedIndividual rdf:about="&Security;FootPrinting">
    <rdf:type rdf:resource="&Security;HostLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#FormFieldManipulation -->

<NamedIndividual rdf:about="&Security;FormFieldManipulation">
    <rdf:type rdf:resource="&Security;ParameterManipulation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Full_access_with_errors -->

<NamedIndividual rdf:about="&Security;Full_access_with_errors">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#HTTPHeaderManipulation -->

<NamedIndividual rdf:about="&Security;HTTPHeaderManipulation">
    <rdf:type rdf:resource="&Security;ParameterManipulation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#High -->

<NamedIndividual rdf:about="&Security;High">
    <rdf:type rdf:resource="&Security;Priority"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Implementation -->

<NamedIndividual rdf:about="&Security;Implementation">
    <rdf:type rdf:resource="&Security;LifeCycle"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#InformationDisclosure -->

<NamedIndividual rdf:about="&Security;InformationDisclosure">
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#InformationGathering -->

<NamedIndividual rdf:about="&Security;InformationGathering">
    <rdf:type rdf:resource="&Security;NetworkLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Integrity -->
```

```
    <NamedIndividual rdf:about="&Security;Integrity">
        <rdf:type rdf:resource="&Security;SecurityConcern"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Intercepting_validator -->

    <NamedIndividual rdf:about="&Security;Intercepting_validator">
        <rdf:type rdf:resource="&Security;SecurityPattern"/>
        <rdfs:comment xml:lang="en">Several well-known attack strategies involve compromising a system by sending requests
with invalid data or malicious code. This entails injection of malicious scripts, SQL statements, XML content and invalid data.
These attacks can be avoided by validating data before use. Because of the constantly changing attack patterns, the data validation
mechanism has to continuously change to prevent against new attacks. Another concern is the freshness of data.Verify the user input
before they are used.</rdfs:comment>
        <Security:hasLayer rdf:resource="&Security;Application"/>
        <Security:hasThreat rdf:resource="&Security;BufferOverflow"/>
        <Security:hasThreat rdf:resource="&Security;Canonicalisation"/>
        <Security:hasThreat rdf:resource="&Security;CrossSiteScripting"/>
        <Security:hasLifecycle rdf:resource="&Security;Design"/>
        <Security:hasSecurityConcerns rdf:resource="&Security;Filtering"/>
        <Security:hasSecurityConcerns rdf:resource="&Security;Integrity"/>
        <Security:hasThreat rdf:resource="&Security;SQLInjection"/>
        <Security:hasDomain rdf:resource="&Security;WebAndJ2EE"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#J2EE -->

    <NamedIndividual rdf:about="&Security;J2EE"/>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#LackOfIndividualAccountability -->

    <NamedIndividual rdf:about="&Security;LackOfIndividualAccountability">
        <rdf:type rdf:resource="&Security;ConfigurationManagement"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Limited_access -->

    <NamedIndividual rdf:about="&Security;Limited_access">
        <rdf:type rdf:resource="&Security;SecurityPattern"/>
        <Security:hasThreat rdf:resource="&Security;AttackerRevealsImplementationDetails"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Low -->

    <NamedIndividual rdf:about="&Security;Low">
        <rdf:type rdf:resource="&Security;Priority"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#LuringAttacks -->

    <NamedIndividual rdf:about="&Security;LuringAttacks">
        <rdf:type rdf:resource="&Security;Authorisation"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#ManInTheMiddle -->

    <NamedIndividual rdf:about="&Security;ManInTheMiddle">
        <rdf:type rdf:resource="&Security;SessionManagement"/>
    </NamedIndividual>


    <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Medium -->

    <NamedIndividual rdf:about="&Security;Medium">
        <rdf:type rdf:resource="&Security;Priority"/>
```

```
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Mitigation -->

<NamedIndividual rdf:about="&Security;Mitigation">
    <rdf:type rdf:resource="&Security;SolutionType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Multilevel_Security -->

<NamedIndividual rdf:about="&Security;Multilevel_Security">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Network -->

<NamedIndividual rdf:about="&Security;Network">
    <rdf:type rdf:resource="&Security;Layer"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#NetworkEavesdropping -->

<NamedIndividual rdf:about="&Security;NetworkEavesdropping">
    <rdf:type rdf:resource="&Security;Authentication"/>
    <rdf:type rdf:resource="&Security;Sensitive_Data"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#NonRepudiation -->

<NamedIndividual rdf:about="&Security;NonRepudiation">
    <rdf:type rdf:resource="&Security;SecurityConcern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#OS -->

<NamedIndividual rdf:about="&Security;OS">
    <rdf:type rdf:resource="&Security;DomainSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#OverPrivilegeProcessAndServiceAccounts -->

<NamedIndividual rdf:about="&Security;OverPrivilegeProcessAndServiceAccounts">
    <rdf:type rdf:resource="&Security;ConfigurationManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#PasswordCracking -->

<NamedIndividual rdf:about="&Security;PasswordCracking">
    <rdf:type rdf:resource="&Security;HostLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#PerimeterSecurity -->

<NamedIndividual rdf:about="&Security;PerimeterSecurity">
    <rdf:type rdf:resource="&Security;AppicationContext"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Policy -->

<NamedIndividual rdf:about="&Security;Policy">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>
```

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#PoorKeyGenerationOrKeyManagement -->

<NamedIndividual rdf:about="&Security;PoorKeyGenerationOrKeyManagement">
    <rdf:type rdf:resource="&Security;Cryptography"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Preventtion -->

<NamedIndividual rdf:about="&Security;Preventtion">
    <rdf:type rdf:resource="&Security;SolutionType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Protected_system -->

<NamedIndividual rdf:about="&Security;Protected_system">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#QueryStringManipulation -->

<NamedIndividual rdf:about="&Security;QueryStringManipulation">
    <rdf:type rdf:resource="&Security;ParameterManipulation"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#RBAC -->

<NamedIndividual rdf:about="&Security;RBAC"/>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Recovery -->

<NamedIndividual rdf:about="&Security;Recovery">
    <rdf:type rdf:resource="&Security;SolutionType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Reference_monitor -->

<NamedIndividual rdf:about="&Security;Reference_monitor">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Replicated_System -->

<NamedIndividual rdf:about="&Security;Replicated_System">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Repudiation -->

<NamedIndividual rdf:about="&Security;Repudiation">
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#RetrievalOfClearTextConfigurationData -->

<NamedIndividual rdf:about="&Security;RetrievalOfClearTextConfigurationData">
    <rdf:type rdf:resource="&Security;ConfigurationManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Role_Based_Access_Control -->
```

```
<NamedIndividual rdf:about="&Security;Role_Based_Access_Control">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SCADA -->

<NamedIndividual rdf:about="&Security;SCADA">
    <rdf:type rdf:resource="&Security;DomainSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SQLInjection -->

<NamedIndividual rdf:about="&Security;SQLInjection">
    <rdf:type rdf:resource="&Security;InputValidation"/>
    <Security:hasSeverityScale>7</Security:hasSeverityScale>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_Communication -->

<NamedIndividual rdf:about="&Security;Secure_Communication">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_Logger -->

<NamedIndividual rdf:about="&Security;Secure_Logger">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_pipe -->

<NamedIndividual rdf:about="&Security;Secure_pipe">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
    <Security:hasThreat rdf:resource="&Security;BruteForceAttack"/>
    <Security:hasThreat rdf:resource="&Security;CookieReplay"/>
    <Security:hasThreat rdf:resource="&Security;CredentialTheft"/>
    <Security:hasThreat rdf:resource="&Security;DictionaryAttack"/>
    <Security:hasThreat rdf:resource="&Security;NetworkEavesdropping"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_proxy -->

<NamedIndividual rdf:about="&Security;Secure_proxy"/>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Secure_service_proxy -->

<NamedIndividual rdf:about="&Security;Secure_service_proxy">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Security_Association -->

<NamedIndividual rdf:about="&Security;Security_Association">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Security_Session -->

<NamedIndividual rdf:about="&Security;Security_Session">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>
```

```
<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Security_context -->

<NamedIndividual rdf:about="&Security;Security_context">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Session -->

<NamedIndividual rdf:about="&Security;Session"/>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SessionHijacking -->

<NamedIndividual rdf:about="&Security;SessionHijacking">
    <rdf:type rdf:resource="&Security;NetworkLevel"/>
    <rdf:type rdf:resource="&Security;SessionManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#SessionReplay -->

<NamedIndividual rdf:about="&Security;SessionReplay">
    <rdf:type rdf:resource="&Security;SessionManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Single_Access_Point -->

<NamedIndividual rdf:about="&Security;Single_Access_Point">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Sniffering -->

<NamedIndividual rdf:about="&Security;Sniffering">
    <rdf:type rdf:resource="&Security;NetworkLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Spoofing -->

<NamedIndividual rdf:about="&Security;Spoofing">
    <rdf:type rdf:resource="&Security;NetworkLevel"/>
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Standby -->

<NamedIndividual rdf:about="&Security;Standby">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Symmetric_Encryption -->

<NamedIndividual rdf:about="&Security;Symmetric_Encryption">
    <rdf:type rdf:resource="&Security;SecurityPattern"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#System -->

<NamedIndividual rdf:about="&Security;System">
    <rdf:type rdf:resource="&Security;Layer"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#Tampering -->
```

```
<NamedIndividual rdf:about="&Security;Tampering">
    <rdf:type rdf:resource="&Security;ThreatType"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UbiquitousComputing -->

<NamedIndividual rdf:about="&Security;UbiquitousComputing">
    <rdf:type rdf:resource="&Security;DomainSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UnauthorisedAccess -->

<NamedIndividual rdf:about="&Security;UnauthorisedAccess">
    <rdf:type rdf:resource="&Security;HostLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UnauthorisedAccessToAdministrationInterface -->

<NamedIndividual rdf:about="&Security;UnauthorisedAccessToAdministrationInterface">
    <rdf:type rdf:resource="&Security;ConfigurationManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UnauthorisedAccessToConfigurationStores -->

<NamedIndividual rdf:about="&Security;UnauthorisedAccessToConfigurationStores">
    <rdf:type rdf:resource="&Security;ConfigurationManagement"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#UserDeniesPerformingAnOperation -->

<NamedIndividual rdf:about="&Security;UserDeniesPerformingAnOperation">
    <rdf:type rdf:resource="&Security;AuditingAndLogging"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#VeryHigh -->

<NamedIndividual rdf:about="&Security;VeryHigh">
    <rdf:type rdf:resource="&Security;Priority"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#VeryLow -->

<NamedIndividual rdf:about="&Security;VeryLow">
    <rdf:type rdf:resource="&Security;Priority"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#VirusesTrojanHorsesAndWorms -->

<NamedIndividual rdf:about="&Security;VirusesTrojanHorsesAndWorms">
    <rdf:type rdf:resource="&Security;HostLevel"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#VoIP -->

<NamedIndividual rdf:about="&Security;VoIP">
    <rdf:type rdf:resource="&Security;ApplicationSpecific"/>
</NamedIndividual>


<!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#WeakOrCustomEncryption -->

<NamedIndividual rdf:about="&Security;WeakOrCustomEncryption">
    <rdf:type rdf:resource="&Security;Cryptography"/>
```

```
        </NamedIndividual>


        <!-- http://www.semanticweb.org/guah1/ontologies/Security.owl#WebAndJ2EE -->

        <NamedIndividual rdf:about="&Security;WebAndJ2EE">
            <rdf:type rdf:resource="&Security;DomainSpecific"/>
        </NamedIndividual>
</rdf:RDF>


<!-- Generated by the OWL API (version 3.4.2) http://owlapi.sourceforge.net -->
```

# Appendix C: List of Publications

[1] H. Guan, W. Chen, L. Liu and H. Yang, "Estimating Security Risk for Web Applications using Security Vectors"，*Journal of Computers*，Vol. 23, No.1, pp. 54-70, Apr. 2012.

[2] H. Guan, W. Chen, N. Huang and H. Yang, "Estimation of Reliability and Cost Relationship (RCR) for Architecture-based Software"，*International Journal of Automation and Computing*, Vol. 7, No.4, pp.603-610, Nov. 2010.

[3] H. Guan, W. Chen, L. Liu and H. Yang, "Environment-driven Threats Elicitation for Web Applications", *5th International KES Conference on Agents and Multi-agent Systems – Technologies and Applications(KES AMASTA'11),* Manchester, UK, pp. 291-300, Jun. 2011.

[4] H. Guan, H. Yang and H. Hakeem, "Reverse Engineering Web Applications for Security Mechanism Enhancement ", *8th IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'14),* Sweden, July, 2014. (in press)

[5] H. Guan and H. Yang, "A Framework for Security Driven Software Evolution", *Proceedings of the 20th International Conference on Automation & Computing*, Bedfordshire, UK, 12-13 Sep. 2014. (in press)

[6] H. Guan, W. Chen and H. Li, "STRIDE-based Risk Assessment for Web Applications", *International Conference on Information Technology for Manufacturing Systems (ITMS'11),* Shanghai, China, pp. 1323-1328, May. 2011.

[7] H. Guan, T. Wang and W. Chen, "Exploring Architecture-Based Software Reliability Allocation Using a Dynamic Programming Algorithm", *2nd International Symposium Computer Science and Computational Technology (ISCSCT'09)*, Huangshan, China, pp.106-109, Dec. 2009.

[8] H. Guan, W. Chen, J. Wang and H. Yang, "An Environment Driven Risk Assessment Model for Web Application", *3rd International Conference on Education Technology and Computer(ICETC'11)*, Changchun, China, Jul. 2011.

[9] H. Guan, W. Chen and H. Yang, "Architecture-based software reliability optimisation allocation", *1st Cross-strait Conference on Software Technology (CSCST'10),* Taibei, Taiwan, pp.95-98, Mar. 2010.

[10] H. Guan, W. Chen and H. Yang, "Study of An Architecture-based Reliability Cost Model for Software", *Proceedings of the 15th International Conference on Automation & Computing (ICAC'09),* Luton, UK, pp.204-208, Sep. 2009.

[11] H. Hakeem, H. Guan and H. Yang, "A Framework of Patterns Applicability in Software Development", *8th IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'14),* Sweden, July, 2014. (in press)

[12] H. Li, H. Guo, H. Guan, Y. Xu and H. Yang, "An approach to evolving business rule-based legacy systems", *2011 International Conference on Computer Applications and Network Security (ICCANS'11),* Maldives, pp.164-168, May. 2011.

[13] J. Wang, J. Ma and H. Guan, "Optimisation and Analysis on S-MAC Protocol based on Priority", *Advanced Materials Research*, Vol.187, pp.73-77, Feb. 2011.

[14] T. Wang, G. Chen and H. Guan, "An Improved Filling Algorithm for Image with Complicated Closed Edges", *Applied Mechanics and Materials*, Vol. 20-23, pp. 894-897, Jan. 2010.

[15] J. Zhang, W. Chen and H. Guan, "Based on Web Application Classification for Threat Research", *Journal of Shenyang University of Chemical Technology*, Vol. 25, No.3, pp. 273-277, Sep. 2011 (in Chinese).